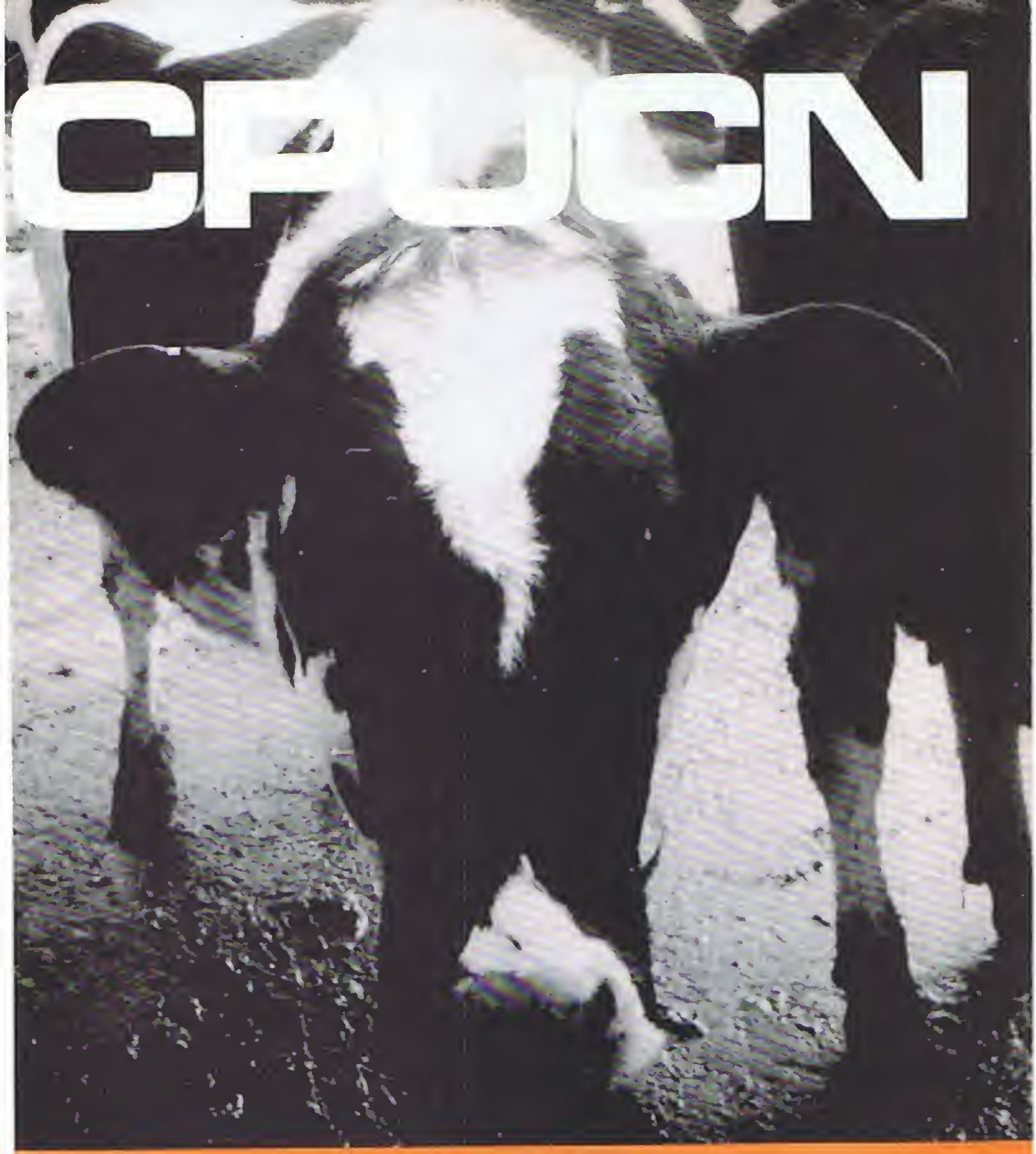
CPUCN



IN THIS ISSUE

- Standard Data Entry Environment
- VIC News
- Farming with a PET
- Parallel to Serial Data Input
- Fast String Handling
- Jump Tables

Volume 3
Issue 3

commodore

CONTENTS

EDITORIAL

Dave Middleton

In the last issue I asked if you were happy with assembler listings being printed in CPUCN, from the letters I received on the subject it would appear that most of you are happy with this situation. CPUCN is however becoming too technical and the beginner is being forgotten to a certain extent. Not everybody is interested in machine code programming and many are struggling to learn BASIC. There is normally a fairly large section called 'BASIC Programming' in the magazine but if you flick through the magazine quickly you will probably miss it this time! So come on readers, send in your BASIC snippets and they will be published. If anybody could take the time to put together an article on using a specific aspect of the BASIC Interpreter, such as trigonometrical functions, arrays or actual programming practices then send them in, they will be much appreciated by the beginners. There have also been a couple of requests for articles on simple disk programming...any offers please!

In this issue there is a very important piece of software, it is called, 'Commodores Data Entry Environment'. This is a machine code program written by Paul Higginbottom which we want software writers to use. The program is available on disk for incorporation in programs, though this offer is only open to recognised software houses by application in writing to Andrew Goltz. Prospective Approved Products Suppliers should note that we will no longer Approve programs for the 8000 series which do not conform to the standard. The program has already been used with success by some current Approved products suppliers.

UK subscribers will find the new Commodore Approved Catalogue included with this issue, at 20 pages it is the biggest catalogue produced by Commodore to date.

COMMODORE NEWS

Dave Middleton

Commodore introduce Letter Quality printer

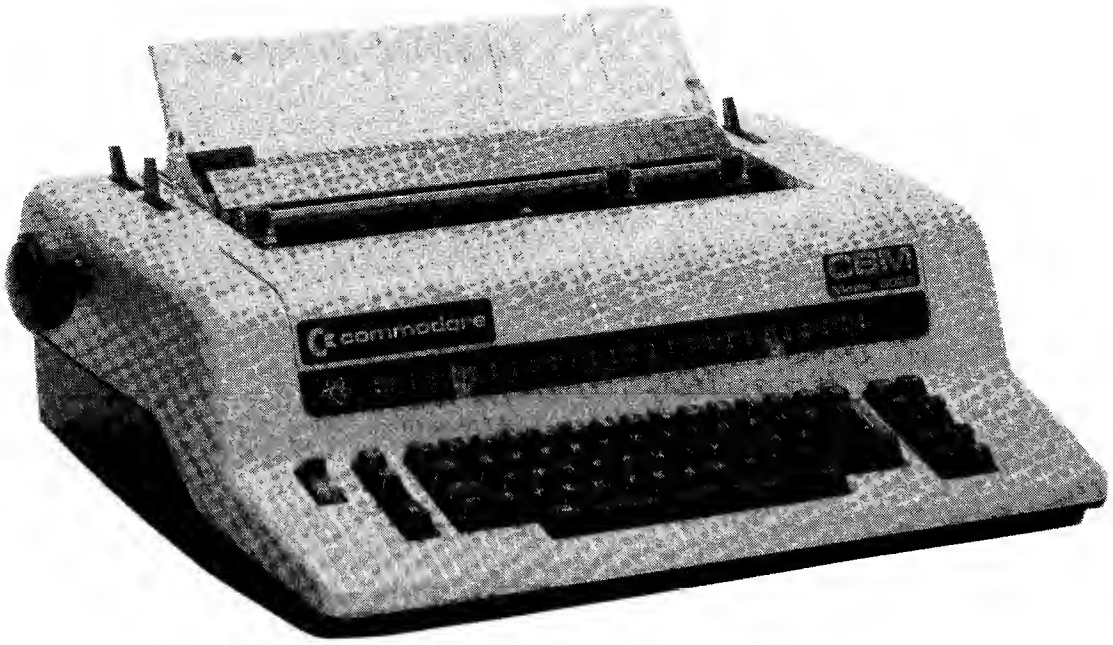
The first batch of 8026 Commodore daisy wheel printers have arrived at Commodore Slough. The 8026, priced at £995.00+VAT is suitable for companies who need to produce letter quality documents from a word processor or accounts package, while retaining the normal functions of a typewriter, there is a second version of the printer called the 8027 which does not have the keyboard and costs £850.00+VAT. The printer has a built in IEEE interface so will attach directly to the PET. The print speed is 16 cps which is good enough for most applications but will be unsuitable for multi-station wordprocessing. The 8026 can be used with 10, 12 or 15 pitch daisy wheels and has 3 line-feed selections.

PET Show 1981

The PET show is going to be held in the West Centre Hotel, London, from the 18-20th of June. Be sure to keep a day free as this years show is going to be bigger and better than last years.

New Approved Products Catalogue

All UK subscribers should have received a copy of the new Approved Products Catalogue with famous names like WordPro4, WordCraft80, VisiCalc, ICI GammaTrol and MuPET all new to the catalogue.

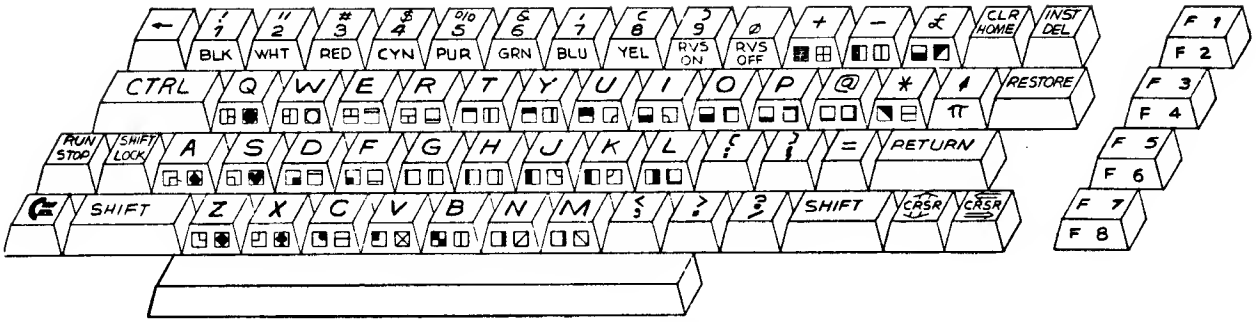


VIC News

The first American specification VIC has arrived in the UK. It is quite a bit different to the picture of the one I showed in CPUCN Issue 3.1, the Japanese character set has been replaced with graphics characters and there is a pound sign on the keyboard, which is fairly revolutionary!

Most of the keys have three functions, upper case is accessible as soon as the VIC is switched on, the right hand graphic is produced with the shift key and the left hand grahic is produced by pressing the shift key and the Commodore logo key simultaneously. Pressing Shift and Commodore together causes the PET to drop into lower case, no messing around with POKE59468.

There is going to be considerable criticism laid against the VIC for having only 22 characters across the screen. However the shape of the pixles is such that they are much wider than deep making the screen layout quite pleasing to the eye, it is however a bit disconcerting seeing a BASIC line spread over 4 screen lines. You can also define a full 255 character graphics set so there are all



sorts of possibilities there for high quality games.

The VIC has been designed to be expandable, this gives it a considerable advantage against a lot of the competitors. It is all very well thinking that you will never have a use which exceeds 8k only to find after a few months of programming that 16k would have been more appropriate. The VIC comes with 5k of RAM which can be expanded in easy stages to 27k. Another idea used on the VIC are plug in ROM cartridges a similar concept to that used by the Atari video computer.

The VIC was designed to be a true personal computer with colour, sound, joysticks, lightpen and low cost peripherals. Expansion peripherals have already been developed and many will be available by the official launch date, these include a Hi-resolution graphics board with a resolution of 176 x 176 points, an RS232C interface board, memory expansion boards and an IEEE PET interface board which will allow all the PET's peripherals to be used with the VIC. The system is not a first generation machine, like the original 8k PET, the designers have had a chance to look around at other manufactures products and combine the best facilities from each into what will be one of the most popular systems available and as an added advantage the VIC has standard PET BASIC, with additions to handle colour, so it will be easy to convert PET programs, both machine code and BASIC, to run on the VIC.

There are eight colours available including black and white. The cursor controls have been moved once again and are now in a more logical position. The RESTORE key is useful in that pressing it and RUN/STOP together causes the PET to reset, clearing all program variables, resetting default values for colour etc, leaving the program ready to run again. The keys to the side of the main keyboard are programmable function keys giving it a unique position amongst home computers. It is a simple matter to be able to assign user defined functions to specific keys calling them up whenever required.

The VIC plugs directly into a standard colour television (you get grey shades on a black and white) so there is no need for expensive monitors.

Price and availability? The big question! Well the price is going to be less than £200.00 and the VIC will be available in the second quarter of this year so start saving your pennies. CPUCN will give the details as soon as they are available so once again, please do not ring your dealer.

Initially we are going to produce VIC News via CPUCN but when the VIC becomes established it will have its own magazine.

Commodore Releases New Structured-BASIC Language, COMAL, as public domain software.

COMAL is a microcomputer language which combines the simplicity of BASIC with the program structure of Pascal. COMAL has had a considerable impact on the educational market - where many experts have been worried that teaching normal BASIC does not induce good program discipline in the student.

COMAL - the COmmon Algorithmic Language was invented by Borge Christensen from Denmark, who is a leading figure in the European education field and the development of COMAL for the PET over the last few months has been achieved with close co-operation with Mr. Christensen. PET COMAL, which is 24k of machine code, was written by Mogens Kjaer of Copenhagen. According to Mr. Christensen, "The PET COMAL is the best implementation of the original concept yet - both in terms of speed and new commands and of course having it on the PET range gives it the maximum possible exposure to schools and colleges in Europe."

Commodore has decided to release the complete program as Public Domain Software - meaning that any PET user is free to copy and use it without payment or royalty. This is despite the substantial development investment by Commodore Electronics Ltd. (CEL).

The benefits of COMAL are readily obvious, even to the novice with such features as IF THEN ELSE ENDIF, REPEAT UNTIL, indented program lines,

long variable names, named procedures and true parameter passing. All this adds up to a language ideally suited to the educational problem solving environment.

Although the lannguage is now finished it will not be released until March, allowing time for field testing and good documentation. In the UK copies will be distributed via Commodore's Education Workshops and User Groups. Please do not apply directly to us.

Free ROM Offer

Some confusion has been created regarding the pricing of BASIC ROMs as the result of my reply to Mr Gordon Brown in the last issue of CPUCN. The following note should clear up any misunderstandings.

Up to 1 January, 1980, to enable users of the original 8K machines to use Commodore's floppy disk unit, BASIC 2.0 ROMs were available free of charge to users purchasing the 2040 floppy disk unit who could prove that their 8K was purchased prior to June 1979 (when BASIC 2.0 machines were first introduced).

After 1 January, 1980 we discontinued the supply of BASIC 2.0 ROM sets as a standard part and they became available as a spare part to special order only. Towards the end of 1980, BASIC 4.0 was introduced. In order to facilitate the interchangeability of software, BASIC 2.0 ROMs were re-introduced as a standard part of our price list, available to the end user at a fixed price of £38.00 exclusive of VAT and installation.

Although the original free ROM offer to 2001 8K PET owners has, in fact, long been discontinued, in view of any confusion which may have arisen as a result of my reply to Mr Gordon Brown (last issue) we are making the following offer, for a limited period only, to PET User Club members.

BASIC 2.0 ROMs will be available, free of charge via your local Commodore dealer, to User Club members purchasing a Commodore 3040 disk unit who can provide their dealer with proof of purchase of an 8K PET, prior to June 1979. This offer will run from 1 December 1980 through to 1 March, 1981.

BUSINESS USERS COLUMN

Barry Miles

More Installation Considerations

In the last column we considered a number of aspects prior to installing your system, and investigated the possible ways of going about buying suitable programs for our purposes.

This article will get further into matters of concern when computerising our business procedures for the first time, in particular accounting procedures. The human aspects are the most important ones in seeking successful operation, so I make no apology for giving them substantial emphasis throughout.

Some people have a phobia against computers. Whether this is a fear of computers in particular or of new experiences in general, is not important. Some enthusiasts are inclined to express surprise at these fears and to say that there is no possible cause for concern. Remember all the science fiction tales, the newspaper reports of computer errors (usually human errors exposed by and/or blamed on the machine, because it cannot answer back!) and also the articles which appear regularly on the theme "Can the machines take over," or "Can computers think?", to realise how reasonable these thoughts are. In any case, the enthusiasts are not immune from being paralysed by indecision when faced with a choice. It is highly amusing and instructive, to put two different commands in a program which require the user to hit a key in order that the program shall move on to its next operation. If you put "hit C to continue", or "hit SPACE to continue", an experinced operator will quite happily comply. If, however you make the command "Hit any key to continue", even experienced programmers will pause for a long while, wondering which key to press!

This brings us, surprisingly enough, to the question of Startrek and Space Invaders! The question is whether we should seek to prevent our staff from playing such games, at least during working hours. Obviously the "work ethic" says that we should but it is not quite so simple as that. We may well find that the familiarity with the computer and enjoyment of the games is just what is needed to remove the reservations about the machine and help to ensure its ready acceptance!

We must bear in mind that the arrival of the computer will be a disruptive event for the staff concerned, and it is very probable that their work will never be the same again! It will be necessary to formalise procedures which may have been left informal up to now.

A matter which is very likely to receive scant attention is the question of Systems Analysis. In the case of large mainframe computer installations, costing hundreds of thousands of pounds, people are quite accustomed to spending a lot of time and expensive effort in the systems analysis phase. During this phase, which precedes the installation of a computer and/or the design of the program, much effort is put into examining what is really required of the system by the various users. The design of output from the programs, the form and content of data required by the various users of that data all have to be examined carefully. Because microcomputing machinery itself is so cheap and the programs bought "off the shelf" are also inexpensive, there is a tendency to expect not to pay out any money on systems research which would perhaps double the cost of the installation; it is usually preferred to try it and see. In fact, of course, this is not very logical. If one's income could suddenly be doubled, it is expected that different ways of spending the extra funds would be experimented with. Similarly, if a way of halving the cost of one's food was found, it would be normal to feel justified in spending more in other ways. Logically therefore, we should be prepared to spend on systems research, having bought the other items at so low a price. To expect this is to ignore the fact that we, unlike mainframe computer users, have not already become accustomed to using such expensive machinery and the idea of further expenditure on expensive research techniques is likely to be unattractive. Furthermore we are likely to be buying packaged programs "off the shelf" and consequently to be unable to influence the design of the program itself.

We need to be aware that computerising any accounting procedure will make it necessary to change the tried and tested methods we have adopted for carrying out the tasks at present and it is advisable to satisfy ourselves that these changes be justified before buying the program. Good program documentation should

indicate what procedures are mandatory for the satisfactory operation of the program, but most are deficient in this regard, and many are devoid of any indication of what is required, apart from a reference to the need to take regular back-up copies of disks. This really is not adequate and the new user of a program is well advised to give considerable thought to matters like, "What will happen to my data if the power fails in the middle of the production of this accounting information", "How can we check that the figures keyed into the machine are accurate?", etc.

Most accounting programs will have some form of audit trail, producing summary totals as a matter of course, but it is up to the user to see that other totals exist, against which to carry out checks. This brings us to the involvement of that much maligned person - the auditor. When should he be involved, if at all?

Many businessmen regard the auditor as a form of licensed nuisance. In fact he is there to see that the owners of the business are not defrauded by the directors and managers, who run the business on the owners' behalf. In doing this, he must satisfy himself whether the accounts show a true and fair view of both the state of the business's affairs at a particular accounting date and a similar view of the results of an accounting period ended on that date. (Even where the owners and the directors are one and the same people, an independent accountant's report will in practice be required by the Inland Revenue, before they will be willing to accept that the reported profits are not understated.) If the internal procedures to guard against fraud and error are inadequate, it will be impossible for the auditor to issue an unqualified report.

Because of his duty to act as a sort of benevolent business bloodhound, the auditor is ideally suited to examining the systems of internal control so as to see that the arrival of the microcomputer has not made fraud and error more likely. Thus it is highly advisable to obtain his advice before the new procedures are set in motion.

All of this may sound somewhat sinister, but there is a serious danger of the breakdown of the system of internal control, by virtue of the fact that the computer makes it possible for one person alone to carry out work previously done by several.
What does this imply? In order to reduce the risk of error, it is customary to set up a system of checks and balances so that we can ensure that figures are always compared with control totals before being taken as correct. In addition, we must assume that there is a risk of one or more of our employees being less honest than we would wish and we subdivide the work in accounting in such a way that fraud is discouraged, by making it impossible to carry out alone. This acts as a major deterrent in that the potential perpetrator will need the co-operation of at least one more member of staff, which is a worrying thing! He or she cannot be sure that the other party will not decide that it is safer to tell management of the approach and obtain the reward of the good and faithful employee!

Once the computer puts one person in charge of the operation of a complete part of the accounting operation, the dangers are substantial, particularly if the person concerned is also the only one around who understands what the machine does and how it does it! Should the person concerned also have programming ability, there is the additional danger of subtle changes being made in the operation of the programs. This suggests that machine-code programs are the safest to use, since they are much more difficult to change.

Calling in the auditor has one rather subtle advantage: staff will be resentful of any implication that they are not fully trustworthy and if internal control procedures are set up in such a way that staff check one another's work, it is much easier to do this if one can rightly explain that "The auditors insist", rather than get into a discussion about trust, or lack of it!

In mainframe computer installations, data preparation is a separate task and often a large department is involved. The microcomputer does not justify anything so grand, but careful thought must be given to the subject nonetheless. A golden rule is that the primary documents on which a computer run is to be made should be batched up together, by someone other than the computer operator and totals prepared of the values of the data being processed. These totals should be compared with those produced by the computer as part of the audit trail, to verify that the data has been entered correctly. The comparison should not be done by the operator alone, for reasons explained above. The size of the batch should be determined logically, based on the characteristics of the task concerned: too small a batch will cause too many labour-intensive checks to be carried out; too large a batch will mean that a single error will take too long to find! There is no rule of thumb for this, it is a matter of individual judgement.

If you can arrange for some batches to be run by another operator, (using a copy of the original program, not accessible to the usual operator, rather than the working copy), you can ensure that the results you are obtaining are those you expect. Your tact and ingenuity may be

taxed to the limit in trying to avoid your staff's feeling that they are being spied upon, although once again the rule of, "When in doubt, blame the auditor", may well come to your rescue. It is worth bearing in mind that the time when you computerise a particular system is the ideal time to review the procedures surrounding the system, with a view to making them more secure, whilst also making them more efficient. One object of consulting the auditor is to avoid a situation where you set up slack, insecure procedures at first, only to be compelled to tighten them up later, to the disgust of staff, some of whom may walk out! It may be helpful to point out to your staff that secure procedures prevent the innocent from being accused of fraud, as well as preventing it, or at least, making it less likely.

All in all, this area calls for the utmost skill in industrial relations, but it is a nettle which must be grasped, if you are to be confident about the security of your financial affairs.

LETTERS

Dear Dave,

Could I make a plea? I have been using an 8032 for 4 months, could CPUCN recognise our needs. Is it not possible to identify whether program listings published are compatible and could you persuade advertisers to do the same. The problem also relates to many other bolt-on goodies, toolkits and the like.

The impression I get from Commodore publications is that the 8032 users should not be interested in computing. They will be provided in the fullness of time and at not inconsiderable cost with 'business' programs and should not require arcade games or to go PEEKing and POKEing around. I hope I am wrong.

Yours sicerely,
J.A.Tanner

Editors reply:

You most certainly are wrong Mr Tanner. While the 8000 series computer is aimed at the business user who sees the machine only as a tool there are many people who are purchasing the machine who have to write their own software so it is obviously to Commodore's benefit if there is as much information available as possible. In issue 3.1 I published a memory map conversion table for BASIC2 to BASIC4 and in the last issue I published an article by Paul Higginbottom after he had converted all the BASIC2 games programs to run on the 4032 which is the 40 column PET with BASIC4. There are only a few differences between BASIC2 and BASIC4 zero page locations. BASIC4 on the 80 column machine is the same as BASIC4 on the 4032, only the screen editors are different.

Dear Dave,

I recently purchased a 3032 CBM and a cassette deck and have noticed a peculiarity in the operation of the machine and would be very grateful if you would advise me whether the problem is known to you.

Basically, I cannot load a program beyond 7FFE as the TIM Monitor appears to ignore all addresses beyond 7FFF. Any machine code programs which are SAVEd from the top of memory will therefore lose their last byte when re-loaded.

A diagnostic program seems to confirm that all the memory and screen is correct and specifically, $7FFF and screen locations respond correctly to PEEK and POKE commands. Automatic relocator programs function ok when loading to the top of memory.

On a different subject, I cannot understand the letter from Jim Butterfield published on page 21 of Issue 3.2, regarding the 'bug' in the Supermon program. The original code in issue 2.4 gives hex $FC(252) at location 1781 and therefore cannot give a value of 26 when PEEKed. Perhaps you could clarify this.

Yours sincerely,
David Pollock

Editor replies:

The cassette SAVE routine is incapable of saving any locations above $7FFF because the checksum for the tape is stored in the high address byte. Also there is a bug in the Monitor in that the last byte to be saved is always lost. Therefore byte $7FFF is always lost when a LOAD is made. With the disk system it is possible to save any area of memory.

The letter from Jim Butterfield giving the bug is concerned with a later version of Supermon which has not been published but has been given away on disk to some user groups. The version has no new commands and generally is only a tidied version of the one published.

Dear Dave,

Before he left Commodore, Mike Whitehead said that something was being planned to assist with the overcrowding of the three spare ROM sockets. I am not sure how far the idea progressed, if at all. No doubt you are aware of the flood of chips claiming space for these three locations and the problems that it causes. Mikes idea was for a board with about nine sockets and this would seem to be a reasonable remedy. The only solutions to date are either to push/pull ROMs from the socket which results in bent pins or to purchase a Spacemaker which is expensive and only switches two devices.

Are there any plans for such a product?

Yours sincerely,
John Nuttall, SUPA

Editors reply:-

Commodore as far as I know has no plans to produce a Spacesaver but we are currently testing a British Spacemaker, for inclusion in the Approved Products scheme, with space for three ROMs, switch selectable and since it is TTL compatible

it should be fairly easy to put onto the User Port thus making it software selectable. The cost is going to be in the £20.00-30.00 range.

Dear Dave,

I have found that it is dangerous to SCRATCH programs from the disk when their blocks have become very scattered. This situation arises when you have a number of programs of widely varying size, and you create new version of them thus needing to scratch the old ones to stop the disk overflowing.

After a few cycles of this kind of operation the blocks will become scattered as free spaces are used. Then, without warning, on scratching a program, you will find that a chain has been broken and surrounding program blocks are no longer linked. You have then lost part of a program which is irretrievable.

Yours sincerely,
W O Murcott

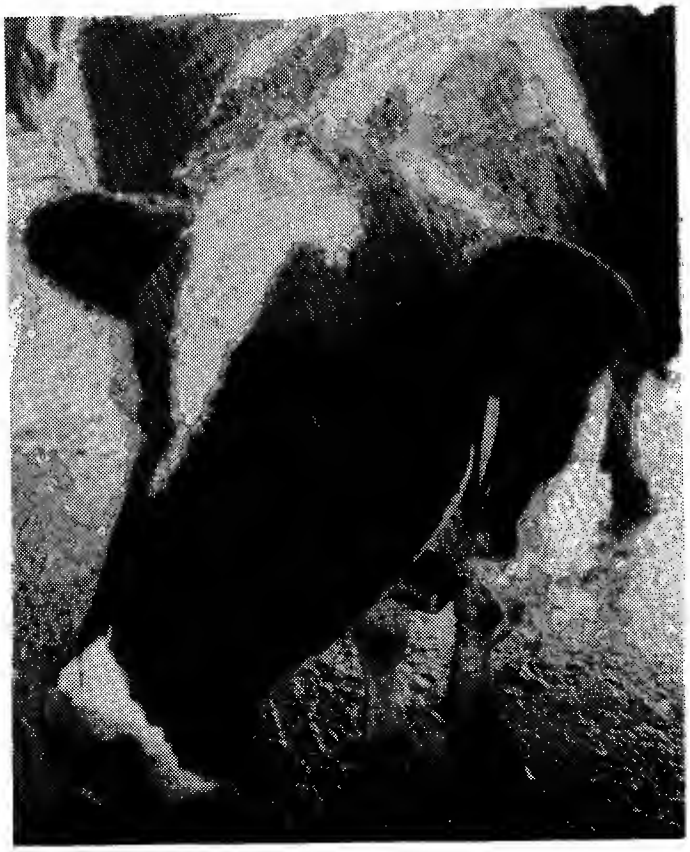
Editors reply:-

Hunting down disk problems is always very difficult if they can not be recreated and only turn up in a special set of circumstances. The above problem occured on a disk I was using during assembler development work. I lost the end of a source file which caused the EDITOR to crash. Luckily I keep a double copy of everything I do on the same disk so was able to recover the failure. It occured to me at the time that the BAM was being updated incorrectly so now when I want to recover large amounts of space by deleting backup copies I tell the disk to VALIDATE after I have scratched all the old versions. This causes the BAM to be recreated, since then I have had no problems.

FARMING

Down on the Farm

Dave Middleton



What's the connection between a graduate micro chip designer, the Commodore PET and Daisy on the front cover?

The answer is Mr. Hotz de Baar. Why would a graduate chip designer want to go into farming? Well it's fairly obvious that Britain is not the centre of the microchip industry! There is Inmos of course but by the time production actually starts most of us will be drawing our old age pensions. Hotz went into farming out of interest and because he has relations who farm. He started work as an assistant manager on a large farm in Hertfordshire and later took over Upthorpe Farm. Initially there were great problems with the farm, couch, a very resiliant form of grass was rampant and there were trace mineral deficiences in the soil. With justifiable pride Hotz pointed out that two millionaires had lost small fortunes trying to make the farm work. Hotz managed to break the stranglehold of the couch grass with a new herbicide and over a few years the farm prospered to its present state. I'm no farmer but it looked very healthy with 1,100 acres of land and a herd of 190 cows. It is in connection with the cows that the PET is concerned. Farming is a very scientific process, it's not just a matter of putting the cow out to pasture and then milking it twice a day.

There is a great deal of information which is necessary for the successful maintenance of the herd. Hotz demonstrated this by saying that a potential £20,000 had been missed due to misinformation before the PET arrived. Being a chip designer Hotz had a great interest in computers and it was obvious to him that a micro could quite easily provide the necessary information. His choice of computer lay either with the Apple or with PET. A farm environment is probably the worst there is for a micro, especially if it is to be used on site by the herdsman. While I was there the temperature in the outhouse was near zero and water was running down the walls. Hotz chose to use the PET because he found his borrowed Apple failed to work below 4°c

The PET is left running all the time for two reasons, the first being that the system is cassette based. This is because disks do not like even small amounts of moisture; in that outhouse there was a lot of water! Thus if the sytem was to be switched off each evening it would require the data base to be reloaded in the morning. With enough space for information on 250 cows it would take a long time for 32k of information to be loaded into RAM.

Mr. O. Hotz de Baar

Leaving the system on all the time means that the information is always present and the herdsman will not be left waiting around for the computer. The second reason for leaving the machine switched on is that it increases the life of the chips. When a chip is working it generates heat and thus expands. When it is switched off the chip will contract. Research shows that the life of the chips is decreased by a factor of 10 when switched on and off.

To many people a computer without disks is a toy but Hotz has obviously thought out his program, called Supercow, very carefully. A program which can handle 250 cows covers approximatly 95% of all herds in this country and larger herds are usually split so it would be possible to have a PET for every herd. It is actually faster than a disk based system because there is no waiting around while the data is read from disk. Most of the program was written in BASIC with machine code subroutines to speed up areas where BASIC is too slow.

So what is Supercow?

The program gives up to date management information in key areas of dairy profitability for single cows, groups or the herd as a whole; this is something which has been either very difficult and time consuming to provide or has been very expensive. (A program called Daisy, running on a mini computer and costing around £10,000 has until now been the only commercially available system).

The actual facilities that the program provides are fairly meaningless to the non-farmer, ie. me, but I was very impressed by the presentation and layout of data giving information ranging from feed allocation to cow servicing (...no not a 5,000 mile service). The individual cow records hold a great deal of information such as yield, feed, date of birth, calving date and veterinary records showing the date and reason for visting.

More details about SuperCow are available from Upthorpe Computers, Upthorpe Farm, Didcote, Oxfordshire.

SuperCow is a Commodore Approved Product and costs £800.00

Computer Programs by Farmers for Farmers

After a somewhat late start, the computer has now arrived as a viable and important part of todays farming operations.

By the mid to late seventies automation in agriculture had already reached an advanced level - particularly in the livestock sector. But in the space of a few months in 1980 many farming activities took the ultimate step forward - physical control of an operation by computer. Pig feeding, dairy parlour operation and environmental control are three examples of this. Although the applications here are increasing all the time cost is restricting them to the larger farming enterprses.

Perhaps more exciting has been the use of the small computer or microcomputer in the farm office - specially programmed to

handle time consuming administrative tasks and provide information vital for decision making.

Someone who quickly saw the opportunities in the latter field was a Norfolk farmer, Brian Gough. Here he runs a mixed farm of some 500 acres where in addition to a varied arable operation he breeds pedigree Hereford cattle.

Almost three years ago a visit to an exhibition on microprocessors in London provided the stimulus for looking at ways that a micro computer could assist his own farming needs. To prove the point he bought one of the early PETs for his own use and with a standard payroll program used his own farm as a guinea pig.

With this running successfully he began to look at specific farming applications, the real reason for buying the computer in the first place. After all, as a farmer with wide experience, who was better qualified to decide the information most vital to the farmer. He first tackled a pedigree cow records program and then a bull weight program.

For a period of twelve months Brian Gough ran the farm computer as a private operation, the last six months working out his own application programs. Early in 1980 he became fully convinced in the future of a low cost computer system for the farm office, so much so that he was prepared to hand over the practical management of the farm to his son and devote himself fully to the new technology.

Brians first task was to decide the type of information required in the different areas of farming, to what extent it tied him to the farm office and the saving in time and accuracy if it was to be handled by computer. He also had to balance the costs of the computer against more conventional outside services such as the farm secretary, a secretarial agency, acountants consultants etc.

Agricultural Services now offer seven applications programs and has a Commodore dealership for selling computer systems. One of the programs, Arable Gross Margins is a Commodore Approved Product.

This program, which will handle up to 100 fields and 10 different farms, it can also be a field diary. The following information is first entered into the computer:-

- the crop history of all the fields over the last five years including the current crops, varieties and acreage.

- fertilizers ordered for the coming year with costs (up to 20 if necessary).

- up to 50 cultivations and costs/acre plus control figures.

Fed with this information the computer will then produce printouts on crop history; fertilizers used and quantity left in store; an analysis of any field or crop; an up to date, complete field diary and the gross margin.

In addition, it is ideal for monitoring individual cultivations, fertilizer or spray chemical applications and it has the facility for updating at the end of the year - thus avoiding the need for entering the crop history more than once.

Brian stresses the simplicity of the system and maintains that a farmer can learn to operate the computer and work the programs after a days training. A total package for a farming operation is likely to be around £3000 made up of the cost of the hardware and £150 upwards for a range of disk programs.

farm computing is still in its infancy but the possibilities are enormous and Brian Gough is achieving success with his motto, "Computer programs prepared for farmers by farmers."

More details about the Arable Gross Margins program may be obtained from Agricultural Computer Services, Roundabout Farm, Thurning, near Melton Constable, Norfolk. The program costs: £350 for a 4000 series machine and £400 for an 8000 series PET.

BASIC PROGRAMMING

Here are a few short routines which you may find useful:-

Keyboard Scan

Paul Higginbottom

This routine demonstrates a method by which many keys can be pressed at once, all of which can be detected. It is ideal for games applications. Change the POKE value in line 20 to lower values for different keyboard rows.

```
10 POKE59411,60
20 POKE59408,9
30 FOR I=1 TO 50: GOSUB100: NEXT
40 POKE59411,61
50 END
100 P=PEEK(59410)
110 FOR J=0 TO 7: IF P AND (2↑J) THEN
    PRINT"1";:NEXT: PRINT"□":RETURN
120 PRINT"0";:NEXT:PRINT"□":RETURN
```

Directory Print

Paul Higginbottom

Have you ever wanted to see the directory without having to load it? Here is a way.

```
10 INPUT"DRIVE NO";D: IF D<>INT(D) OR D<0
   OR D>1 THEN 10
20 N$=CHR$(0): H=256: OPEN1,8,0,"$"+MID$(
   STR$(D),2)
30 GET#1,A$,A$
40 GET#1,A$,A$,A$,B$: IFST THEN CLOSE1:
   END
50 PRINTASC(A$+N$)+ASC(B$+N$)*H;
60 GET#1,A$:IF A$<>""THEN PRINTA$;: GOTO6
   0
70 PRINT: GOTO40
```

Version Test

John Collins

With 4 versions of PET and 3 disk systems it is quite often necessary for a program to decide if it can work on a machine, here is a routine which gives the answers:-

```
100 REM VERSION TEST FOR PET AND DISK
110 REM BY: JOHN COLLINS
120 REM PET 2001=0: 3032=1: 4032=2:8032=3
130 A=PEEK(57345): TP=0: IF A THEN TP=1:
    IF A AND 1 THEN TP=3: IF A AND 4
    THEN TP=2
140 REM
150 REM DISK 3040=1: 4040=2: 8050=3
160 REM
170 OPEN15,8,15: PRINT#15,"M-R"CHR$(255)
    CHR$(255): GET#15,A$: CLOSE15
190 A=ASC(A$): TD=1: IF A AND 16 THEN TD
    =3: IF A AND 1 THEN TD=2
200 PRINT TP,TD
```

Duration of a key press

Bob Sharpe

In **Volume3 issue1** a short routine was published for finding the duration of a key press, here is another simpler way of doing this:-

```
10 X=PEEK(151):IFX=YTHENPRINTA$;:GOTO10
20 PRINT:Y=X:GETA$:GOTO10
```

String Operations in BASIC

A.G. Price: Principal Lecturer: Department of Mathematics, Liverpool Polytechnic

Although string comparisons and assignments are fairly efficiently handled in BASIC, this is at the expense of a time consuming process known as garbage collection, which is activated to recover the space occupied by dead strings whenever BASIC runs out of data space. Assignment and string expressions are the biggest creators of dead space and wasted time and this article presents three short machine code routines which can speed up string processing - exchange strings, create strings and insert strings.

My experiments in this area were performed as a result of the article by Nick Marcopoulos on the sorting of strings. Running his sort program showed that for a small number of strings,

direct comparison of strings and assignment is about 28% faster then the indexed method described, which comes into its own as soon as garbage collection becomes necessary. Now in practice strings are indexed anyway: a string variable contains the address of the actual string and its length. Two strings can therefore be interchanged by swapping the addresses contained in their variables. This does not use up string space and cannot invoke garbage collection. Note that it is not safe to assign a string variable by copying its address, this would result in two variables pointing to the same string which garbage collection would not be able to cope with. Exchange of strings is a 46 byte machine code routine and if it is placed in the second cassette buffer the sort program may be altered as follows:-

```
6 W=826: DIMW$,J,K,I: INPUT"NO. OF ITEMS"
  ;N: N=N-1: DIM A$(N)
35 FOR I=Ø TO N: PRINTA$(I): NEXT: PRINT:
   PRINT
40 FOR I=1 TO N: IF A$(I-1)>A$(I) THEN
   PRINT
55 FOR I=M+K TO N STEPK: SYSW,W$,A$(I)
56 FOR J=I-K TO M STEP-K: IF A$(J)>W$
   THEN SYSW,A$(J+K),A$(J):NEXT
58 SYSW,A$(J+K),W$: NEXT I: IF K>1 GOTO53
```

The above modifications should be made to the sort programme by Nick Marcopoulos published in CPUCN Volume3 Issue1.

Note that X(N) is no longer needed. Timing tests show that this version is 12% faster than the original and saves the space occupied by the array X(N) whilst preserving the feature of garbage collection not being invoked.

It is important to note that an assignment eg. W$=A$(I) should be replaced by a call of exchange only when the value on the right hand side is no longer required as it is changed (to the old value of the left hand side).

Timing of various methods of accessing machine code in BASIC

The usual way of accessing machine code from BASIC is by means of the SYS instruction, followed by the absolute address in RAM of the code to be executed. This is commonly written as an integer constant, assuming that the position of the code is known when the BASIC program is written, eg. the second cassette buffer 826-1017. However, BASIC is faster at looking up variables than converting constants and the SYS command runs faster if the start address of the machine code program is assigned as a variable which is then used as the argument of the SYS command. (Lines 1-3 of table). Variables have been pre-defined so that A is the first variable, Z the 26th which can be conveniently done using the DIM statement, eg.

```
DIM A,B$,C,D,E,F,G,H,I,J,K,L,M,N,O,P,
    Q,R,S,T,U,V,W,X,Y,Z$
```

Which creates any variable not previously existing and assigns Ø to it.

The table demonstrates that SYS <variable> is faster as long as the variable is not later than 80th in the variable list. The constant Ø is processed rather faster than all others and so are variables with the value Ø.

Lines 4-6 show that SYSØ is much faster, beaten only by SYS<variable> where the variable is not later than 12th in the variable list.

Location Ø-2 would be loaded with the unconditional jump to the start of the machine code routine, which would add a few microseconds to the execution time. A more serious effect is that it would prevent USR routines from being executed.

An incidental effect of keeping the addresses of machine code routines in variables is that they can be set up when the program is run, eg. whilst packing several small routines into the cassette buffer, or whilst loading routines to the top of RAM and adjusting the top of memory pointers accordingly. (BASIC2 locations 52,53).

Another method of dealing with short (less than 255 bytes) machine code routines written in relocatable code (ie no JMPs or JSRs to other areas of the same code) which can be suprisingly efficient is to transfer them to a string variable and then execute them via a short (27 byte) routine which must be in a fixed location. Lines 7-10 of the table show that this is still faster than SYS<constant> as long as the sum of the positions of the two variables is less than 58 but cannot compete for speed with an indirect SYS via location Ø.

Machine code can be assigned to a string by a BASIC routine similar to the following:-

```
1ØØØ B$="": FOR I=1 TO N: READ J:
     B$=B$+CHR$(J): NEXT
```

or by using the insert routine described below.

It is interesting to compare the use of SYS with the alternative method of extending BASIC by patching machine code programs into BASIC's fetch character (CHRGET at $ØØ7Ø) routine. Such patched routines are entered each time BASIC fetches a character from the program (and at other times also) and have to execute code to determine where they should perform their function. This imposes an

overhead on all BASIC operations: DOS SUPPORT, for instance adds 0.033 milliseconds each time a character is fetched and adds about 7.5% to the run time of Nick Marcopoulos' SORT program, even though it is not used. The SYS command has a larger overhead but it is incurred only when the SYS command is actually executed.

Whilst running timing tests on SORT, it becomes apparent that the process of constructing random test data is quite time consuming. The usual process is to use the concatenate operator in a FOR loop. This generates all the intermediate strings on the way to the final string and takes a minimum of (9+5*K) milliseconds, where K is the length of the string. A similar problem occurs when using a string as a record and attempting to change a value which is stored as a substring ("field") within the 'record' string eg.

```
5 REM CREATE A 10-CHARACTER STRING OF
  SPACES(61 MSEC).
10 D$="": FOR J=1 TO 10: D$=D$+" ": NEXTJ
15 REM INSERT A 5-CHARACTER FIELD AT
   CHARACTERS 3-7(61 MSEC).
20 D$=LEFT$(D$,2)+"*****"+RIGHT$(D$,3)
```

These processes can be speeded up by a 117 byte machine code routine CREATE & INSERT STRINGS. This will create a string variable of any length in a fixed time of about 7.1 milliseconds. eg.:-

```
10 SYS XXX,A$,100: REM CREATE 100
   CHARACTER STRING A$
```

The contents of the string will be rubbish at this point. The same routine, with a third string parameter, will insert the third string (variable, constant or expression) into the first string variable starting at the character position given by the second parameter eg.

```
10 SYSXXX,A$,3,"*****": REM INSERT *****
   INTO A$ 3-7
```

This takes between 7 and 8 milliseconds for constant data.
These routines, together with 16bit PEEK & POKE are included in the BASIC program shown. The operative lines for transferring code and recording its start address in variables are 60020, 60030 and 60060 but the various routines in lines 60120-60240 can be used as required. Lines 60010, 60050 and line 60070 may require some explanation.

BASIC lines are linked by an address at the head of each line which points to the head of the next line. End of program is indicated by an address after the last line set to zero. Variables start at an address held in locations 42/3 which set to the location after the end of program whenever an edit or load is performed. While a program is running locations 58/9 contain the address of the last byte of the previous statement. Line 60010 stores the address of the end of line 60000 in A: line 60040 resets start of variables to commence where the text of the statement 60010 was and sets start of arrays and end of arrays to simulate CLR. Now up to nine variables at 7 bytes each may be created without disturbing the BASIC program as long as no attempt is made to obey line 60010 again. Line 60050 makes the link address at the head of line 60010 (start of variables minus 4) point to the start of line 60050 so that all the space between the start of line 60010 and 60050 can be used as data space giving room for another 24 variables. This statement can be repeated as necessary, to turn program into data space. The statement POKE(USR(42)-3): in line 60070 turns the start of line 60010 into the end of program marker so that all the program data statements, initial dialogue creation of variables and so on can be wiped out to make space for data once it has been executed. Try running the program, then LIST and print out the locations of start of variables (42/3), start of arrays (44/5) and end of arrays (46/7).

Timing Tables, SYS command

Command		Time (Millisec)
SYS826:		4.36
SYSA:	[A=826]	1.33
SYSZ:	[Z=826]	2.24
SYS0:		1.36
SYSA:	[A=0]	0.89
SYSZ:	[Z=0]	1.83
SYSA,B$:	[A=826]	2.04
SYSA,B$:	[A=0]	1.60
SYSZ,B$:	[Z=826]	3.93
SYSZ,Z$:	[Z=0]	3.49

Each increase of 1 in the position of a variable adds 0.038 milliseconds to the execution time.

The machine code obeyed consisted of the single instruction RTS (return).

```
60000 GOSUB60010:STOP
60010 A=PEEK(58)+256*PEEK(59)+5:REM SAVE ADDR. THIS REM MAKES SPACE FOR 9 VARIABLES.
60020 W=834:GOSUB60170:REM LOAD 16-BIT POKE TO BUFFER 41
60030 SYS634,1,W:GOSUB60170:REM 16-BIT PEEK TO FOLLOW
60040 SYS634,42,A:SYS834,44,USR(42):SYS834,46,USR(42):REM RESET START,END OF VARS.
60050 SYS834,USR(42)-4,USR(58)+1:REM MAKE LINES 60020-60040 VARS. SPACE
60060 W=826:A=W:GOSUB60170:B=W:GOSUB60170:C=W:GOSUB60170
60070 POKE(USR(42)-3),0:RETURN:REM RESET END-OF-PROGRAM & EXIT.
60080 REM-----------------------------------------------
60090 REM SYSA,EXPR$...OBEY STRING AS CODE
60100 REM SYSB,VAR$,VAR$: EXCHANGE STRINGS
60110 REM SYSC,VAR$,EXPR: CREATE STRING VAR$ OF LENGTH EXPR
60120 REM SYSC,VAR1$,EXPR,VAR2$: INSERT VAR2$ INTO VAR1$ AT CHARACTXR EXPX
60130 REM-----------------------------------------------
60140 READZ:FORY=1TOZ:READX:NEXTY:REM SKIP CODE.
60150 REM-----------------------------------------------
60160 REM LOAD CODE TO CURRENT POSITION W.W IS UPDATED TO FIRST FREE SPACE.
60170 READZ:FOXY=WTOW+Z-1:READX:POXXY,X:NEXTY:W=W+Z:RETURN
60180 REM-----------------------------------------------
60190 REM LOAD CODS TO STRING $$.THIS SHOULD NOT BE USED BEFORE LOAD TO SIMEM.
60200 READZ:Z$="":FORY=1TOZ:READX:Z$=Z$+CHR$(X):NEXTY:RETURN
60210 REM-----------------------------------------------
60220 REM LOAD CODS TO SIMEM AND ADJUST SIMEM.DESTROYS $TXINGS.USES 16-BIT DPB.
60230 READZ:W=DBX(52)-Z:SYS634,52,W:SYS834,58,W:SYS834,48,W:REM ADJUST SIMEM,$TSING
60240 FORY=WTOW+Z-1:READX:POKEY,X:NEXTY:RETURN:REM W=ADDRESS OF CODE.
60250 REM-----------------------------------------------
60260 DATA41:REM 16-BIT POKE. BYS XXX,$XPR1,EXPR2:EXPR1-ADDRESS,$XPR2=VALUE.
```

```
60270 DATA22,246,205: REM JSR $CDF6  ;CHECK FOR COMMA
60280 DATA32,129,204: REM JSR $CC65  ;EVALUATE NUMERIC EXPRESSION
60290 DATA32,210,214: REM JSR $D6D2  ;CHECK,16-BIT INTEGER...
60300 DATA165,16:     REM LDA $12    ;..TO $61/2,A/Y,..
60310 DATA72:         REM PHA        ;..$11/2...
60320 DATA165,17:     REM LDA $11    ;..SAVED ON STACE.
60330 DATA72:         REM PHA
60340 DATA32,240,205: REM JSR $CDF6  ;CHECK FOR COMMA
60350 DATA22,139,204: REM JSR $CC8E  ;EVALDATE NUMERIC EXPRESSION
60360 DATA22,210,214: REM JSR $D6D2  ;CHECK,16-BIT INTEGER...
60370 DATA170:        REM TAX        ;..TO A/Y -> E/Y
60380 DATA104:        REM PLA
60390 DATA123,17:     REM STA $11    ;RESTORE ADDRESS...
60400 DATA164:        REM PLA
60410 DATA123,16:     REM STA $12    ;FROM STACK.
60420 DATA152:        REM TYA
60430 DATA160,0:      REM LDY #0
60440 DATA145,17:     REM STA ($11),Y;POKE L/S HALF
60450 DATA200:        REM INY
60460 DATA128:        REM TXA
60470 DATA145,17:     REM STA ($11),Y;POKE M/S HALF
60480 DATA96:         REM RTS        ;EXIT.
60490 REM-------------------------------------------------
60500 DATA26:REM 16-BIT PEEK. USR(<ADDRESS>).PLANT START ADDRESS IN $1/2.
60510 DATA165,16:     REM LDA $12    ;SAVE $11/2...
60520 DATA72:         REM PHA
60530 DATA165,17:     REM LDA $11    ;...ON STACK.
60540 DATA72:         REM PHA
60550 DATA32,210,214: REM JSR $D6D2  ;CONVERT F.P. ACC. TO 16 BITS...
60560 DATA166,1:      REM LDT $1     ;..IN $01/2,$11/2,A/Y
60570 DATA177,17:     REM LDA ($11),Y;M/S HALF OF VALUE...
60580 DATA170:        REM TAX        ;..TO X,..
60590 DATA138:        REM DEY
60600 DATA177,17:     REM LDA ($11),Y;..L/S HALF...
60610 DATA168:        REM TAT        ;..TO T
60620 DATA104:        REM PLA        ;RESTORE $11/2...
60630 DATA133,17:     REM STA $11
60640 DATA164:        REM PLA
60650 DATA122,16:     REM STA $12    ;..FROM STACK
60660 DATA138:        REM TXA        ;VALUE IN A/T
60670 DATA76,186,210: REM JMP $D26D  ;CONVERT TO F.P. AND RETURN.
60680 REM-------------------------------------------------
60690 DATA27:REM CODE TO DEEK A STRING AS MACHINE CODE. SYSXXX,EXPR$.27 BYTES.
60700 DATA32,240,205: REM JSR $CDF6  ;TEST FOR COMMA
60710 DATA32,150,204: REM JSR $CC9F  ;INPUT EXPRESSION
60720 DATA36,7:       REM BIT $7     ;TEST TYPE OF EEPRESSION
60730 DATA48,3:       REM BMI *+5    ;STRING
60740 DATA76,3,200:   REM JMP $CE63  ;NUMERIC= ERROR
60750 REM             ---------
60760 DATA160,2:      REM LDY #2
60770 DATA177,87:     REM LDA $81,T  ;ADDRESS OF STRING...
60780 DATA133,86:     REM STA $56
60790 DATA136:        REM DEY
60800 DATA177,87:     REM LDA $81,T
60810 DATA122,87:     REM STA $57    ;...TO $57/8
60820 DATA160,87,0:   REM JMP ($57)  ;EXECUTE STRING
60830 REM-------------------------------------------------
60840 DATA48:REM CODE TO EXCHANGE STRINGS.SYS XXX,A6,$8.48 BYTES.
60850 DATA22,240,205: REM JSR $CDF6  ;CHECK FOR FIRST COMMA
60860 DATA22,188,207: REM JSR $CF8D  ;FETCH VARIABLE FROM BASIC
60870 DATA165,60:     REM LDA $44    ;SAVE ADDRESS OF VALUE...
60880 DATA72:         REM PHA        ;...OF STRING VARIABLE...
60890 DATA165,88:     REM LDA $45
60900 DATA72:         REM PHA        ;...ON STACK.
60910 DATA22,144,204: REM JSR $CC88  ;CHECK, STRING VARIABLE
60920 DATA32,240,205: REM JSR $CDF6  ;CHECK FOR SECOND COMMA
60930 DATA32,188,207: REM JSR $CF8D  ;FETCH SECOND VARIABLE
60940 DATA32,144,204: REM JSR $CC88  ;CHECK, STRING VARIABLE
60950 DATA104:        REM PLA
60960 DATA122,85:     REM STA $5F    ;RETRIEVE FIRST VALUE ADDRESS...
60970 DATA104:        REM PLA
60980 DATA122,94:     REM STA $5E    ;...FROM STACK
60990 DATA168,2:      REM LDY #2     ;COUNT TO SWAP 2 BYTES
61000 DATA177,68:     REM LDA ($44),Y;<^
61010 DATA170:        REM TAX        ; ^ SAVE FIRST
61020 DATA177,94:     REM LDA ($5E),Y; ^
61030 DATA145,66:     REM STA ($44),Y; ^ REPLACE BY SECOND
61040 DATA128:        REM TXA        ; ^ RETRIEVE FIRST
61050 DATA145,94:     REM STA ($5E),Y; ^ SAVE AT SECOND
61060 DATA126:        REM DEY        ; ^
61070 DATA10,242:     REM BPL *-11   ;>^ DO NEET BYTE
61080 DATA96:         REM RTS        ;END OF EXCHANGE.
```

```
61090 REM-------------------------------------------------
61100 DATA117:REM CODE TO INSERT A STRING EXPRESSION INTO A STRING VARIABLE.
61110 REM SYSXXX,VAR$,POS,EXPR$  E.G.
61120 REM A$="**********":$$="+++++":SYS828,A$,2,$8:7A$
61130 REM **++++***
61140 REM CREATE A STRING VARIABLE (SUBSISS CONTENTS).
61150 REM STSXXX,VAR6,EXPR  E.G.
61160 REM STS826,A6,10:7LEN(A$)
61170 REM  10
61160 DATA32,240,205: REM JSR $CDF6  ;TEST FOR COMMA
61190 DATA32,109,207: REM JSR $CF6D  ;FETCH VARIABLE ID
61200 DATA123,70:     REM STA $46
61210 DATA122,71:     REM STY $47    ;SAVE ADDRESS OF VALUE
61220 DATA192,222:    REM CPY #$DE   ;TEST, TIS7
61230 DATA200,3:      REM BNE *+5    ;NO
61240 DATA76,3,200:   REM JMP $CE82  ;YES-"SYNTAX ERROR"
61250 REM             ---------
61260 DATA32,144,204: REM JSR $CC88  ;CHECK ID IS STRING
61270 DATA32,240,205: REM JSR $CDF6  ;CHECK,COMMA AFTER ID
61280 DATA22,120,214: REM JSR $D87B  ;FETCH START POSITION TO E. $02
61290 DATA22,115,0:   REM JSR $78    ;TERMINATOR
61300 DATA201,44:     REM CMP #$2C   ;..COMMA?
61310 DATA240,8:      REM BEQ *+10   ;YES-DO INSERT
61320 DATA120:        REM TXA        ;NO-DO CREATE
61330 DATA32,87,211:  REM JSR $D357  ;CREATE STRING EFACS
61340 DATA22,111,201: REM JSR $C98F  ;PLANT VALUE IN ID
61350 DATA96:         REM RTS        ;END OF CREATE.
61360 REM             ---------
61370 DATA202:        REM DEX        ;INSERT.
61380 DATA138:        REM TXA
61390 DATA72:         REM PHA        ;SAVE ON STACK
61400 DATA32,167,204: REM JSR $CCA7  ;EVALUATE EXPRESSION
61410 DATA32,144,204: REM JSR $CC88  ;CHECK,STRING EXPRESSION
61420 DATA104:        REM PLA
61430 DATA133,17:     REM STA $11    ;START POSITION IN TARGET STRING
61440 DATA160,0:      REM LDY #0
61450 DATA5$:         REM SEC
61460 DATA177,70:     REM LDA (646),T;LENGTH OF TARGET STRING...
61470 DATA241,97:     REM SBC ($61),T;..-LENGTH OF INSERT STRING
61480 DATA176,5:      REM BCS *+7    ;INSERT TOO LONG?
61490 DATA162,176:    REM LDX #$B0   ;<
61500 DATA76,67,165:  REM JMP $C357  ; ^ TE$-"STRING TOO LONG"
61510 REM             ---------
61520 DOTA167,17:     REM CMP $11    ; ^
61530 DATA144,247:    REM BCC *-7    ;>^ INSERT+START TOO LONG
61540 DATA177,97:     REM LDA ($61),T
61550 DATA170:        REM TAX        ;X=LENGTH OF INSERT
61560 DATA200:        REM INY
61570 DATA24:         REM CLC
61580 DATA177,70:     REM LDA($46),Y
61590 DATA101,17:     REM ADC $11    ;$4D/E=ADDR. OF TARGET+START...
61600 DATA133,77:     REM STA $4D    ;$4D/E=ADDR. OF TARGET+START...
61610 DATA177,97:     REM LDA ($61),Y
61620 DATA133,95:     REM STA $5F    ;..$5F/60=ADDRESS OF INSERT
61630 DATA200:        REM INY
61640 DATA160,0:      REM LDA #0
61650 DATA112,70:     REM ADC ($46),Y
61660 DATA122,70:     REM STA $46
61670 DATA177,87:     REM LDA ($61),Y
61680 DATA133,88:     REM STA $58
61690 DATA165,87:     REM LDA $61    ;CHECK,IF ZERO. IS TOP...
61700 DATA164,88:     REM LDY $62    ;..OF VALUE STACK...
61710 DATA32,161,212: REM JSR $D5B5  ;..DELETE IT.
61720 DATA120:        REM TXA
61730 DATA168:        REM TAY
61740 DATA240,8:      REM BEQ *+10   ;INSERT OF ZERO LENGTH
61750 DATA120:        REM DEY
61760 DATA177,95:     REM LDA ($5F),Y;< COPY INSERT...
61770 DATA145,77:     REM STA ($4D),Y; ^...TO TARGET
61780 DATA130:        REM DEY        ; ^
61790 DATA16,248:     REM BPL *-5    ;>^
61800 DATA96:         REM RTS        ;END OF INSERT.
```

MACHINE CODE

Jump Tables in Assembler Code

Mike Gross-Niklaus

1. What is a jump table.

Quite often when writing a program, you come to a point where you need to branch along one of the several different paths, depending on the value of a particular variable. In BASIC, this need is catered for by the ON..GOTO statement. Suppose you have a line as follows:-

```
1020 ON N GOTO 2000, 3000, 4000, 2500
```

Then if N=1 the program will branch to line 2000. If N=2 the branch will be to 3000 and if N=4 then it will branch to 2500. (In Pascal a similar facility is provided with the CASE statement.) You could call the list of line numbers following the ON..GOTO statement a jump table.

2. Using JMP indirect.

The 6502 instruction set includes a jump indirect. This is a three byte instruction with the second and third bytes referring to the address where the jump destination is stored. Thus if location $7FFE contains $1A and $7FFE contains $30 a jump indirect instruction: JMP ($7FFE) will cause a jump to location $301A.

3. Using JMP () with a jump table.

If you set up a sequence of jump addresses somewhere in memory ie. a jump table, you can index this table with either the X or Y register. By copying the low and high bytes of the indexed table entry into a predefined location and then doing a JMP () using that location you can cause the program to branch to one of a number of possible points depending on what is in the index register.

A neat way to set out the table is to put all the high bytes together and all the low bytes together in another area. This allows the index register to point to jump destination N when it contains a value of N.

4. Example using jump indirect.

In the first example of assembler listing, "INDIRECT.S", the high bytes of the table are set up in locations $0300 to $0302 (JMPHI) and the low bytes in locations $0303 to $0305, (JMPLO). Locations $033A to $0348 contain the coding which sets up the jump and is followed by three dummy routines in $0349, $034F and $0355 so you can check the procedure really works. The * in the label declerations (DUM1 =*) tells the assembler to make a note of the current address thus removing the need for the user to have to worry about where to put code. If the user wants to control the actual position of the code than an address can be given eg. DUM1 =$033A

To do the check a BASIC test program is used. It asks you for a value 0 to 2 and POKEs it to be executed. (A little later on in this article a different method of doing jumps is explained using RTS. This BASIC program can be used to to test either method.)

The value POKEd into $00 (POINT) by the BASIC program is loaded into the X register as an index to the jump table. The high and low bytes of the jump so indexed are copied into a pair of locations, (INDIR and INDIR+1). The JMP (INDIR) will then cause a jump to whichever location has been set up in INDIR and INDIR+1.

For example, suppose you replied '1' to the BASIC program INPUT prompt in line 1020 then a value of 1 will be put into POINT and copied to the X register. Using the table values shown in the example, first a value of $03 will be copied into INDIR+1 then a value of $4F to INDIR. JMP (INDIR) will cause the jump to locations $034F.

5. Shortening the code using RTS instead of JMP ()

Particularly in dedicated applications where to keep the cost down, RAM storage will be limited, there is often a need to economise on program space. ie. the shorter the program the better in some cases.

It is possible to achieve the same effect as JMP () using the properties of RTS, while cutting the number of bytes required for the set up routine from 15 to 11.

The method relies on the fact that the RTS instruction pulls its return address from the stack in page one of memory. However there is a slight 'gotcher'. The normal use of the RTS is in conjunction with a JSR instruction. When a JSR is obeyed the high byte of the program counter is first pushed onto the stack and then the low byte. After a subroutine has been obeyed, execution

should pick up at the location following the JSR address so the operation of RTS is to pull the address from the stack and put them into the program counter register and then increments the program counter by one. This ensures that the next instruction is fetched from the location following the second byte of the JSR address.

To use RTS as a jump instruction you have to push the high and low bytes of an address onto the stack and then do an RTS. Because the RTS increments the address by 1 it is neccessary to store a location 1 byte short of where you really want to go. Using the dummy routine examples the required locations on the stack would be $0344, $034A or $0350 rather than $0345, $034B, $0351 (see JMPRTS2.S) respectively.

6. An example of using RTS with a jump table.

The second assembler code listing, JMPRTS2.S, shows an equivalent procedure to INDIRECT.S using RTS with a jump table. The table is set up in two parts as before but this time locations in the table are 1 less than the location required. The set up coding is in $033A to $0344, a saving of 4 bytes. The dummy routines remain unchanged, except that now the labels are made to point to the previous instruction (*-1) and the same BASIC program can be used to check out the method.

The BASIC program gets a value 0 to 2 and POKEs it into POINT. From here the assembler program copies it to the X register. The high byte of the location is pushed onto the stack followed by the low byte-1. The RTS is then executed, placing the previously stacked address into the program counter which is then incremented by one. The next instruction is then fetched using the program counter as normal.

For example, supposing you replied to the BASIC program INPUT prompt in line 1020 with a 1. The X register is loaded with this value. Using the X register as a table pointer first a value of $03 then a value of $4A are copied from the table and pushed onto the stack. The RTS causes the address $034A to be pulled from the stack and placed into the program counter. The register is then incremented by 1 to give $034B. The program then continues processing from location $034B.

7. Conclusion

Both of these techniques appear frequently in assembly code programs. For example, have a look at a disassembly of EXTRAMON. You will see the second method used to decode the various extra monitor commands and cause a jump to the appropriate subroutine.

```
READY.


 100 REM************************
 110 REM* IND/RTS.B             *
 120 REM* MIKE GROSS-NIKLAUS    *
 130 REM* 17/11/80              *
 140 REM************************
 150 REM
 160 INPUT"ENTER 0,1 OR 2";N
 170 IF N<0 OR N>2 THEN 160
 180 POKE0,N: SYS826
 190 GOTO160


JMPRTS2.S.....PAGE 0001>1
LINE# LOC   CODE        LINE


0001  0000              SCREEN =$8000
0002  0000              POINT  =$00
0003  0000                                  ;
0004  0000                      *=$0300     ;
0005  0300                                  ;
0006  0300  03          JMPBI  .BYTE >DUM1, >DUM2, >DUM3
0006  0301  03
0006  0302  03
0007  0303  44          JMPLO  .BYTE <DUM1, <DUM2, <DUM3
0007  0304  4A
0007  0305  50
0008  0306                                  ;
0009  0306                      * =$033A    ;
0010  033A                                  ;
0011  033A  A6 00        SETJMP LDX POINT     ;AS POKED
0012  033C  BD 00 03            LDA JMPBI,X  ;BI BYTE OF JUMP
0013  033F  48                  PHA
0014  0340  BD 03 03            LDA JMPLO,X  ;LO BYTE OF JUMP-1
0015  0343  48                  PHA
0016  0344  60                  RTS          ;MAKE A DUMMY RETURN
0017  0345                                  ;
0018  0345              DUM1   =*-1
0019  0345  A9 01              LDA #01      ;PUT AN 'A' ON THE SCREEN
0020  0347  8D 00 80            STA SCREEN
0021  034A  60                  RTS
0022  034B                                  ;
0023  034B              DUM2   =*-1
0024  034B  A9 02              LDA #02      ;PUT A 'B' ON THE SCREEN
0025  034D  8D 00 80            STA SCREEN
0026  0350  60                  RTS
0027  0351                                  ;
0028  0351              DUM3   =*-1
0029  0351  A9 03              LDA #03      ;PUT A 'C' ON THE SCREEN
0030  0353  8D 00 80            STA SCREEN
0031  0356  60                  RTS
0032  0357                                  ;
0033  0357                      .END


ERRORS = 0000


SYMBOL TABLE

SYMBOL VALUE
 DUM1    0344    DUM2    034A    DUM3    0350    JMPBI   0300
 JMPLO   0303    POINT   0000    SCREEN  8000    SETJMP  033A

END OF ASSEMBLY


INDIRECT.S.....PAGE 0001
LINE# LOC   CODE        LINE


0001  0000              SCREEN =$8000
0002  0000              POINT  =$00
0003  0000              INDIR  =$01
0004  0000                                  ;
0005  0000                                  ;
0006  0000                      *= $0300
0007  0300  03          JMPHI  .BYTE >DUM1, >DUM2, >DUM3
0007  0301  03
0007  0302  03
0008  0303  49          JMPLO  .BYTE <DUM1, <DUM2, <DUM3
0008  0304  4F
0008  0305  55
0009  0306                                  ;
0010  0306                      *= $033A    ;
0011  033A                                  ;
0012  033A  A6 00        SETJMP LDX POINT     ;AS POKED
0013  033C  BD 00 03            LDA JMPHI,X  ;HI BYTE OF JUMP
0014  033F  85 02                STA INDIR+1  ;INDIRECT HI
0015  0341  BD 03 03            LDA JMPLO,X  ;LO BYTE OF JUMP
0016  0344  85 01                STA INDIR    ;INDIRECT LO
0017  0346  6C 81 00            JMP (INDIR)  ;DO THE JUMP
0018  0349                                  ;
0019  0349              DUM1   =*
0020  0349  A9 01              LDA #01      ;PUT AN 'A' ON THE
0021  034B  8D 00 80            STA SCREEN   ;TOP OF THE SCREEN
0022  034E  60                  RTS          ;RETURN TO BASIC
0023  034F                                  ;
0024  034F              DUM2   =*
0025  034F  A9 02              LDA #02      ;PUT A  'B' ON THE SCREEN
0026  0351  8D 00 80            STA SCREEN
0027  0354  60                  RTS
0028  0355                                  ;
0029  0355              DUM3   =*
0030  0355  A9 03              LDA #03      ;PUT A 'C' ON THE SCREEN
0031  0357  8D 00 80            STA SCREEN
0032  035A  60                  RTS
0033  035B                                  ;
0034  035B                      .END


ERRORS = 0000

SYMBOL TABLE

SYMBOL VALUE
 DUM1    0349    DUM2    034F    DUM3    0355    INDIR   0001
 JMPHI   0300    JMPLO   0303    POINT   0000    SCREEN  8000
 SETJMP  033A
END OF ASSEMBLY
```

PET as an IEEE-488 Logic Analyser

Jim Butterfield

If you would like to see what is happening on the GPIB and you can borrow an extra PET and IEEE interface cable this program will help.

It shows the current status of four of the GPIB control lines, plus a log of the last nine characters transmitted on the bus.

The four control lines are NRFD, NDAC, DAV and EOI. It would have been nice to show ATN too but I could not fit this in, it is detected in a rather odd way in the PET so that putting it in would have been somewhat tricky for this simple program.

The last nine characters are shown in 'screen format'. This means that you will have to do a little translation work to sort out what some of them mean. On the other hand, it allows you to see characters which would normally not be printed. A carriage return shows up as a lower case 'm'; this is a little confusing at the start but you will quickly get used to it and it is handy to see everything that goes through.

I had hoped to show which characters were accompanied by the EOI signal. It turned out that timing was critical, the bus works very fast and that adding this feature would cut down the number of displayed characters from nine to five. I opted for the bigger count and dropped the EOI log feature.

The high speed of the bus makes it difficult to watch the control lines in real time. When the 'active' PET is exchanging information with the disk or printer everything is happening very fast and the 'logic analyser' PET will show an amazing flurry of activity on the control lines. Only when when the activity stops or hangs will you be able to see the lines in their static conditions.

You may use the program to chase down real GPIB problems or just to gain an insight into how the bus works. Either way, it will come in handy if you can borrow that extra PET.

```
30 POKE59468,14: PRINT"□ DAV NRFD NDAC  E
   OI" :PRINT"  ↑    ↑    ↑    ↑M"
40 PRINT"=123456789=M"
```

```
READY.

B*
     PC  IRQ  SR AC XR YR SP
.;  0401 E62E 32 04 5E 00 F4
.
.:  04B0 46 B1 78 AD 12 E8 C9 EF
.:  04B8 D0 02 58 60 AC 10 E8 AD
.:  04C0 40 E8 AE 20 E8 29 C1 C5
.:  04C8 B2 D0 11 98 29 40 0A 49
.:  04D0 A0 C5 B3 F0 DE 85 B3 8D
.:  04D8 61 80 D0 D7 85 B2 29 80
.:  04E0 49 A0 8D 52 80 10 1D A4
.:  04E8 B1 30 1B 85 B1 85 B2 A0
.:  04F0 00 B9 A2 80 99 A1 80 C8
.:  04F8 C0 08 D0 F5 8A 49 FF 8D
.:  0500 A9 80 B0 AF 85 B1 A5 B2
.:  0508 29 40 0A 49 A0 8D 57 80
.:  0510 A5 B2 29 01 4A 6A 49 A0
.:  0518 8D 5C 80 D0 00 00 00 3F
.?
```

```
1000          ;IEEE WATCH BY JIM BUTTERFIELD
1010          ;
1020          *=$4B0
1030          DFLAG   =$B1
1040          DNNSAV  =$B2
1050          EOISAV  =$B3
1060 START    LSR DFLAG
1070          SEI
1080 MAIN     LDA $E812
1090          CMP #$EF
1100          BNE CONT
1110          CLI
1120          RTS
1130 CONT     LDY $E810     ;EOI
1140          LDA $E840     ;DAV, NRFD, NDAC
1150          LDX $E820     ;DATA
1160          AND #$C1      ;EXTRACT BITS
1170          CMP DNNSAV
1180          BNE DNN
1190          TYA
1200          AND #$40          ;EXTRACT EOI
1210          ASL A
1220          EOR #$A0
1230 EOI      CMP EOISAV
1240          BEQ MAIN
1250          STA EOISAV
1260          STA $8061
1270          BNE MAIN
1280 ;ACTIVITY SEEN - UPDATE SCREEN
1290 DNN      STA DNNSAV
1300          AND #$80
1310          EOR #$A0
1320          STA $8052
1330          BPL NDAV          ;NO DAV SEEN
1340          LDY DFLAG
1350          BMI DCONT    ;DAV SEEN BEFORE
1360          STA DFLAG
1370          STA DNNSAV
1380          LDY #0
1390 SCROL    LDA $80A2,Y
1400          STA $80A1,Y
1410          INY
1420          CPY #8
1430          BNE SCROL
1440          TXA
1450          EOR #$FF
1460          STA $80A9
1470          BCS MAIN
1480 NDAV     STA DFLAG
1490 DCONT    LDA DNNSAV
1500          AND #$40          ;NRFD
1510          ASL A
1520          EOR #$A0
1530          STA $8057
1540          LDA DNNSAV
1550          AND #$1           ;NDAC
1560          LSR A
1570          ROR A
1580          EOR #$A0
1590          STA $805C
1600          BNE MAIN
1610          .END
```

Data Input to the Commodore PET via a Parallel to Serial Converter

A C R Strutt and K W Hobbs

On page 110 of 'The PET Revealed', an electronic circuit and appropriate driver software are offered to enable parallel data to be serialised and input into the PET computer. Unfortunately, a system using this circuit and driver will not work. The reasons are as follows.

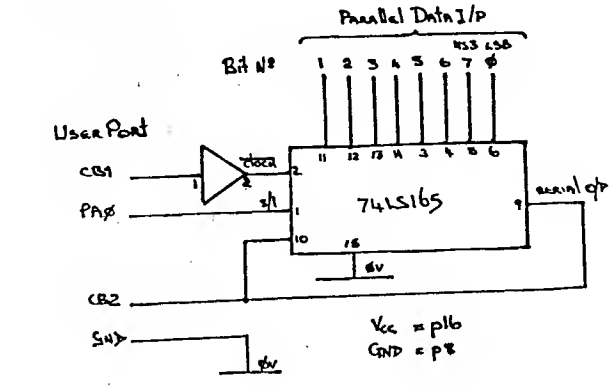
The signals generated by the PET computer come from the Versatile Interface Adaptor (VIA) integrated ciruit, part number 6522. Investigation of the manufacturers data sheet shows that for a peripheral line to output a signal, it must first be configured as an output and then the desired signal levels must be written as ones and zeros to that selected output bit. Thus to load parallel data into an external shift register (ESR) the load/serial mode line of the register must be pulsed to a low logic level (zero). To achieve this PA0 must be configured as an output and then a low logic level written to it, followed by a return to a high logic level (one).

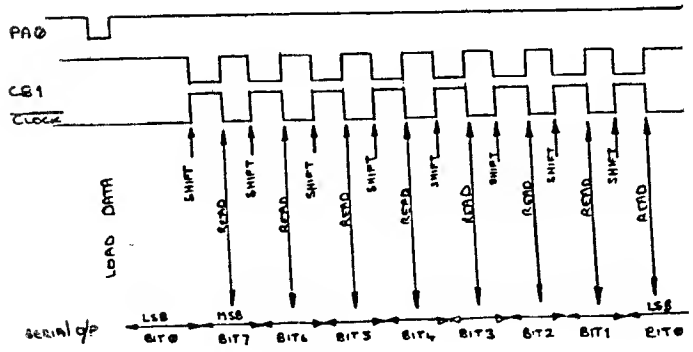
The ESR in the published circuit is a 74LS166: this shift register requires that the clock line is pulsed synchronously with the load/shift mode line being low. In this application, such a requirement adds unnecessary difficulties. The solution is to change the shift register for a 74LS165 which is identical to the 74LS166 but loads parallel data when the load/serial mode line is low only. Having loaded the parallel data, the next problem is how to get the data from the ESR into the shift register (SR) within the VIA. The 6522 data sheet reveals that when using the internal shift register/timer 2 combination in the 'shift in under control of T2' mode, the shifting

operation is triggered by writing to, or reading from the internal shift register (SR). Executing this read or write causes eight clock pulses to be output from VIA CB1 and data to be clocked out of the ESR into the SR via CB2. Further examination of the operation of the VIA shift register shows that data must be valid prior to the rising edge of the CB1 clock pulses. This means the ESR must be clocked by the falling edge of the CB1 clock pulse. This requires that an inverter is included in the CB1 line, since the ESR also requires a rising edge to shift.

Implementing the above raises the problem that the ESR is immediately clocked at the beginning of the shift in cycle, thus losing the most signifiacnt data bit (MSB). To overcome this, the serial output of the ESR is wired to its serial input and the parallel input data lines are all moved one place to the left: the left-most bit (the least significant data bit LSB) is now connected to the MSB input bit of the external shift register (ESR).
When the falling edge of the clock from CB1 appears inverted as a rising edge at the clock input to ESR, the parallel loaded LSB is reloaded via the serial input into the now vacated LSB of the ESR. The serial output now presents to CB2 the MSB of the parallel loaded data in a static state ready for reading by the VIA. The shifting and reading is now continued for the remainder of the clock pulses: the last bit to be read is the reloaded LSB.

Single Byte 8 Bit Parallel to Serial Converter





PA0 Strobes the data (bits 0-7) into eight master slave flip flops. Bit 0 immediatly appears at pin 9 serial O/P.

CB1 goes to a '0' logic level and on the negative edge the shift register is clocked such that bit 7 appears on the serial O/P and bit 0 is loaded into the serial I/P.

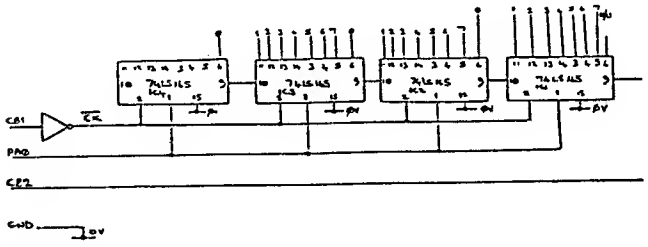
CB1 goes to a '1' level and on the positive edge the data is read. This cycle is repeated until the eighth clock pulse where bit 0 is read and the cycle is completed.

BASIC program to run the 'single byte' input

```
100 POKE 59459,1: REM SET DDR FOR
    PA0 AS AN OUTPUT
110 POKE 59471,1: REM PA0=1
120 POKE 59471,0: REM PA0=0
130 POKE 59471,1: REM PA0=1
140 POKE 59464,64: REM SET TIMER T2
    TO CLOCK AT 1/64
150 POKE 59467,PEEK(59467)AND227OR4:
    REM SET VIA SR UNDER CONTROL OF T2
160 POKE 59466,0: REM DUMMY WRITE TO
    TRIGGER VIA SR
170 WAIT 59469,4: REM WAIT FOR SR
    INTERRUPT IE VIA SR FULL
180 X=PEEK(59466): PRINT X: REM DISPLAY
    CONTENTS OF VIA SR
```

Lines 100-130 allow the strobe pulse on PA0 to be generated, this loads the data into the external SR (74LS165).
Lines 140-150 set up the VIA to clock in data under the control of T2 at a rate of 1/64 ie. 1 MHz/64 (Note the value can be changed to suit any requirements).
Line 160 Triggers the VIA SR to start clocking the external data.
Line 170 waits for all the data to be clocked in.
Line 180 Displays on the PET screen the data.

Three Byte Input



Operation is similar to the single byte circuit except the serial O/P is not fed back to the I/P. To allow for this one extra IC is required to offset the loss of data on pin 6 of the first device.

PA0 strobes all the data into the shift registers and CB1 clocks in the data from pin 9 of IC1. As each bit is shifted and read, the bits in all locations are shifted one bit to the right so that they all eventually appear at pin 9 IC1.

BASIC program to run the 'three byte' Input

```
100 POKE 59459,1
110 POKE 59471,1
120 POKE 59471,0
130 POKE 59471,1
140 POKE 59464,64
150 POKE 59467,PEEK(59467)AND227OR64
160 POKE 59466,0
170 WAIT 59469,4
180 X=PEEK(59466)
190 WAIT 59469,4
200 Y=PEEK(59466)
210 WAIT 59469,4
220 Z=PEEK(59466)
230 PRINTX,Y,Z
```

Lines 100-180 are the same as the single byte program.
The internal VIA SR is triggered when writing or reading the shift register.
Line160 triggers the internal shift register to collect the first byte.
Line180 reads the shift register and automatically triggers the shift register to collect the next byte, hence line 160 is only required for the first byte.

The Commodore Standard User Data Entry Environment

Paul Higginbottom

Commodore Software Department

This standard was established by Commodore, in cooperation with a number of principal software houses. It was designed to provide both end users and vendors with a consistent method of operation for all 8000 series business programs, whether from Commodore or Approved Product suppliers. The main problem is the data entry environment. Commodore decided to make available a utility that would provide programmers with a data entry editor, that could be used in conjunction with both machine code & BASIC programs. If a programmer feels that he/she could not incorporate the actual CBM utility into a particular program, then Commodore would like to see him/her emulate the standard user data entry environment using his own software routines. Obviously, there may be circumstances where not all the standards are appropriate and the use of a subset would be sufficient. In case of difficulty please contact Andrew Goltz with respect to Commodore Approved Software and Mike Gross Niklaus with regard to any software being written specifically for Commodore.

The Standards

01) Every Screen Display should be titled, or indicated as to its purpose. We suggest that the top one or more lines are used.

02) A status line should exist on the bottom of the screen. This informs the user what he/she has to do next (e.g 'PRESS 'C' TO CONTINUE'), any errors that occur (e.g 'ILLEGAL STOCK CODE'), what the computer is doing (e.g 'SORT IN PROGRESS').

03) A shifted return should be used to accept a whole screenful of information.

04) The 'C' key should be used to proceed off one display onto the next.

The rest of the standards are concerned with data entry where more than one field of information is to be entered onto a screen display.

05) The editor should allow full editing of each field (insert, delete, cursor left, cursor right)

06) The HME key should put the cursor at position one in field one.

07) The CLR HME key should either i) empty the fields, ii) reset the fields to their default responses (default values), or iii) reset the fields to their contents when the 'record' was displayed. We would recommend option iii) because if the record was blank when the editor was entered, the fields will be cleared, but if for example a CHANGE was being done to a record, and the user has made a complete mess of the changes, then the record returns to its original state.

08) The RETURN key will either cause a field to be validated immediately, or move the cursor onto the next field on the screen. I will explain a little further on what happens if the user is on the last field on the screen.

09) The CRSR UP, CRSR DOWN keys will allow movement back to the previous, and on to the next field, respectively.

10) The RUN/STOP key should provide an abort/help facility. This could either return the user to a menu/command mode, or provide 'help text'.

11) A shifted RETURN should provide immediate exit from the editor, to a screen accept/reject option.

12) If RETURN or CRSR DOWN were struck when in the last field down the screen, or a SHIFT RETURN had been pressed from any other field, then the 'SCREEN ACCEPT MODE' is entered. This is an ACCEPT/REJECT point for the user, rejection may return the user to a menu or allow them to re-edit the data. We would suggest that the user is allowed to re-edit the data if a rejection is made, because if they decide not to enter any data, then the RUN/STOP key could be pressed. When in the screen accept mode,

a SHIFTED RETURN should be used to accept the data on the screen, or any of CRSR UP, HME, or CLR HME should be allowed, to let them re-enter the editor. RUN/STOP may also be allowed as a help/abort function.

The Commodore Data Entry Editor

This utility was designed around the rules specified above. This routine is all in machine code, and resides at $7B00 (changes to the editor will be dicussed in a moment). The editor uses one byte ($00) to communicate with BASIC as an error code when the editor returns to BASIC. It also uses one array (SC$(n) - screen). The array is for the editor to return the data entered, back to BASIC. A different array may be used. BASIC programs should lower the top limit of usable RAM below the editor to prevent variables walking all over it. This can be done by modifying the contents of 52 and 53 ($34, $35). e.g.:-

```
POKE52,0:POKE53,123:CLR
```

Delimiters

The editor requires that delimiters are put on the screen (in any way you like), that show where the fields are. The delimiters the editor currently uses, are less than ('<'), and greater than ('>') symbols. The space between these delimiters is assumed by the editor to be where the field is. The editor requires that BASIC sets up each array element length to the corresponding field length on the screen. This is so that the editor doesn't have to create strings, but merely replace the current contents of each array element with the data read off the screen. The editor gets the length of each array element and reads that many characters from the start of the corresponding field on the screen (1st field = SC$(0), 2nd field = SC$(1) etc.). The editor will keep reading fields off the screen, until either i) there are no more elements in the array, or ii) the end of the screen has been reached.

What does the error code do?

Byte $00 contains an error code which can be 0,1, or 2. The meaning is as follows:-

PEEK(0)=0 : The data was accepted. An attempt was made to fill the array.

PEEK(0)=1 : A CLR HME had been pressed. No attempt was made to fill the array.

PEEK(0)=2 : The RUN/STOP key was pressed. No attempt was made to fill the array. We would suggest in this case, that an 'ARE YOU SURE ?' message appears in this case.

The editor was designed always to give the user a second chance, but at the same time give the user flexibility without hindrance.

Modifying the Editor

The editor enables repeat functions on 40 column machines.

The RUN/STOP key should be disabled before entering the editor, because if it is pressed, during the editor's operation, then the return to BASIC is so fast, that it is picked up by BASIC, and BREAK IN LINE XXX occurs. Disabling in BASIC2.0 and 4.0 can be done with the statement POKE144,PEEK(144)+3, and then re-enabled with POKE144,PEEK(144)-3.

The memory location of the editor is $7B00, but this may be changed by altering the origin of the assembler source file. The *=$7B00 is the statement to change. If you only know the decimal equivalent, then the Assembler will understand that too (e.g *=32000).

The source listings are fully commented for easy changes. However you may wish to call me on specific points. This is not an open invitation for you to ask me to create tailor made versions for you because that is not my job. Therefore, if you are not familiar with machine code and you wish to make fairly major changes, then you may have to use the services of a software house, familiar with the Commodore Assembler, to make the changes you require.

```
1000 ;*********************************
1010 ;
1020 ;    80 COLUMN & BASIC 4.0
1030 ;
1040 ;*********************************
1050 ;
1060 ENDSCR=34687
1070 SCREEN=32848
1080 LINLEN=80
1090 PRINT=$E202
1100 MSGPN=22
1110 ;
1120 *=$7B00
1130 .LIB   SC1.S
1140 FINISH RTS
1150 .END
```

```
READY.

 1000 ;*******************************
 1010 ;
 1020 ;    40 COLUMN & BASIC 2.0
 1030 ;
 1040 ;*******************************
 1050 ;
 1060 ENDSCR=33727
 1070 SCREEN=32808
 1080 LINLEN=40
 1090 PRINT=$E3D8
 1100 MSGPN=2
 1110 IRQCDE=$E62E
 1120 ;
 1130 ;SET UP REPEAT FOR 40 COLUMN
 1140 ;
 1150 *=$7B00
 1160        LDA $90
 1170        STA IRQLO
 1180        LDA $91
 1190        STA IRQHI
 1200        SEI
 1210        LDA #<REPCDE
 1220        STA $90
 1230        LDA #>REPCDE
 1240        STA $91
 1250        CLI
 1260 .LIB   SCl.S
 1265 .LIB   RPT.S
 1270 FINISH SEI             ;RESET IRQ
 1280        LDA IRQLO       ;TO SAVED VALUE
 1290        STA $90
 1300        LDA IRQHI
 1310        STA $91
 1320        CLI
 1330        RTS       ;AND RETURN TO BASIC

READY.


$0$0

READY.
```

```
 1000 ;*******************************
 1010 ;
 1020 ;    40 COLUMN & BASIC 4.0
 1030 ;
 1040 ;*******************************
 1050 ;
 1060 ENDSCR=33727
 1070 SCREEN=32808
 1080 LINLEN=40
 1090 PRINT=$E202
 1100 MSGPN=2
 1110 IRQCDE=$E455
 1120 ;
 1130 ;SET UP REPEAT FOR 40 COLUMN
 1140 ;
 1150 *=$7B00
 1160        LDA $90
 1170        STA IRQLO
 1180        LDA $91
 1190        STA IRQHI
 1200        SEI
 1210        LDA #<REPCDE
 1220        STA $90
 1230        LDA #>REPCDE
 1240        STA $91
 1250        CLI
 1260 .LIB   SCl.S
 1265 .LIB   RPT.S
 1270 FINISH SEI             ;RESET IRQ
 1280        LDA IRQLO       ;TO SAVED VALUE
 1290        STA $90
 1300        LDA IRQHI
 1310        STA $91
 1320        CLI
 1330        RTS       ;AND RETURN TO BASIC
 1340        .END

READY.

READY.
```

```
1000 ;***************************************
1010 ;* REPEAT CODE FOR 40 COLUMN PET.      *
1020 ;***************************************
1030 ;
1040 DELAY  .BYT 0          ;REPEAT DELAY
1050 REPCTR .BYT 0          ;REPEAT COUNTER
1060 IRQLO  .BYT 0          ;STORE FOR INTERRUPT
1070 IRQHI  .BYT 0          ;REQUEST VECTOR
1080 ;
1090 REPCDE LDA 151         ;GET KEYPRESS
1100        CMP #255        ;NOTHING ?
1110        BME REPEAT      ;NO - TEST REPEAT
1120        LDA #0          ;ZEROISE REPEAT COUNTER
1130        STA REPCTR
1140        LDA #16         ;SET REPEAT DELAY TO 16
1150        STA DELAY
1160        BME IRQ         ;AND GO ON
1170 REPEAT INC REPCTR      ;INCREMENT COUNTER
1180        LDA DELAY       ;GET REPEAT DELAY
1190        CMP REPCTR      ;HAS COUNTER REACHED THIS ?
1200        BME IRQ         ;NO - GO ON
1210        LDA #4          ;YES - SET THE LIMIT TO BE SHORTER
1220        STA DELAY
1230        LDA #255        ;FOOL OPERATING SYSTEM
1240        STA 151         ;TO THINK 'NO KEY PRESSED'
1250        LDA #0          ;ZEROISE COUNTER AGAIN
1260        STA REPCTR
1270 IRQ    JMP (IRQLO)     ;AND JUMP TO USUAL IRQ
1280 ;
1290 .EMD

READY.

READY.
```

```
1000 ;***************************************
1010 ;*                                     *
1020 ;*     FIXED SCREEN IMPUT ROUTINE      *
1030 ;*        BY PAUL HIGGINBOTTOM         *
1040 ;*   (COMMODORE SOFTWARE DEPARTMENT)   *
1050 ;*                                     *
1060 ;***************************************
1070 ;
1080 ;USES '<' AS OPEMING DELIMITER,
1090 ;AND  '>' AS CLOSING DELIMITER
1100 ;OF A FIELD.
1110 ;
1120 ;LABELS USED:-
1130 ;
1140 GET    =$FFE4           ;DOES A 'GET'
1150 POINTR =1               ;GENERAL POINTER LO=1,HI=2
1160 ERRFLG =0               ;ERROR FLAG / GENERAL FLAG
1170 OPENDL =60              ;PEEK CODE OF '<' SYMBOL
1180 CLSEDL =62              ;PEEK CODE OF '>' SYMBOL
1190 CRSRLO =196             ;- START ADDRESS OF LINE
1200 CRSRHI =197             ;  WHICH CURSOR IS ON
1210 CRSRPN =198             ;CURSOR POSITION (0-LINLEN)
1220 CRSRLN =216             ;CURRENT LIME CURSOR IS ON.
1230 ARRTAB =$2C             ;POINTER TO START OF VAR.
1240 ARREND =$2E             ;POINTER TO END OF VAR.
1250 ARRPTR =$23             ;POINTER TO ARRAY
1260 STRPTR =$25             ;POINTER TO DATA FROM ARRAY
1270 NUMINS =220             ;NUMBER OF INSERTS LEFT
1280 QTEFLG =205             ;QUOTE FLAG
1290 PIA    =155             ;PIA KEYSWITCH FOR RUNSTOP/RVS
1300 ;
1310 ;ILLEGAL OR CONTROL KEYS :-
1320 ;
1330 HME    =19              ;HOME CURSOR
1340 CLR    =147             ;CLR SCREEN & HOME CURSOR
1350 RVS    =18              ;REVERSE FIELD ON
1360 OFFRVS =146             ;REVERSE FIELD OFF
1370 CU     =145             ;CURSOR UP
1380 CD     =17              ;CURSOR DOWN
1390 CR     =29              ;CURSOR RIGHT
1400 CL     =157             ;CURSOR LEFT
1410 RTN    =13              ;RETURN
1420 DEL    =20              ;DELETE
1430 IMS    =148             ;IMSERT
1440 ESC    =27              ;ESCAPE
1450 TAB    =9               ;TAB CHARACTER
1460 SHFRTN =141             ;SHIFTED RETURN
1470 SHFTAB =137             ;SHIFTED TAB
```

```
1480 SHPESC =155              ;SHIFTED ESCAPE
1490 RUNSTP =3                ;RUN/STOP
1500 ;
1510 ;REQUIRES THAT:-
1520 ;EACH FIELD ON THE SCREEN HAS:-
1530 ; A '<' AT THE START AND.....
1540 ; A '>' AT THE END OF EACH PIELD
1550 ;
1560 ;*************************************
1570 ;* MAIN ENTRY POINT. RESETS POINTERS, *
1580 ;* THEN SCANS FOR A PIELD. IP ONE IS  *
1590 ;* FOUND, THEN THE CODE GOES ON, ELSE *
1600 ;* SCREEN ACCEPT MODE IS ENTERED.     *
1610 ;*************************************
1620 ;
1630 BEGIN  JSR RESET         ;SET POINTERS TO START
1640 MAIN   JSR UNPLSH        ;UNHIGHLIGHT CURSOR
1650        JSR SCANFD        ;SCAN POR A PIELD
1660        BCC OK            ;FOUND ONE - GO ON
1670 ;
1680 ACCEPT JSR UNPLSH        ;UNHILIGHT CURRENT POSITION
1690        LDA #<ACCMSG      ;DO SCREEN ACCEPT
1700        LDY #>ACCMSG      ;SET A,Y TO MESSAGE
1710        JSR PRTSTR        ;AND PRINT IT
1720 KYLOOP JSR GET           ;DO A 'GET'
1730        BEQ KYLOOP        ;NOTHING PRESSED - GO BACK
1740 KY     CMP #SHFRTN       ;IS IT A SHIFTED RETURN ?
1750        BNE KY0           ;NO - GO ON
1760        JMP GETSTR        ;YES - GO ON TO READ SCREEN
1770 KY0    CMP #HME          ;HOME ?
1780        BNE KY1           ;NO - GO ON
1790        JSR ERASE         ;ERASE MESSAGE
1800        JMP BEGIN         ;GO DO IT
1810 KY1    CMP #CU           ;CURSOR UP ?
1820        BNE KY2           ;NO - GO ON
1830        LDA #1            ;SET FLAG POR ONE SCAN ONLY
1840        STA ERRFLG        ;
1850        JSR ERASE         ;ERASE MESSAGE
1860        JSR BD1           ;SCAN BACK TO BOTTOM FIELD
1870        BCS ACCEPT        ;IP NOT POUND - GO BACK
1880        JMP OK            ;AND CARRY ON
1890 KY2    CMP #CLR          ;CLEAR SCREEN ?
1900        BNE KY3           ;NO - GO ON
1910        JSR ERASE         ;ERASE MESSAGE
1920        LDA #CLR          ;RESET CLR CHAR
1930        JMP OK4           ;GO DO IT
1940 KY3    CMP #RUNSTP       ;RUN/STOP ?
1950        BNE KYLOOP        ;NO - GO BACK
1960        LDA #2            ;SET ERROR FLAG TO 2
1970        STA ERRFLG
1980        LDA #$PP          ;NULLIFY RUNSTOP PLAG
1990        STA PIA
2000        JMP PINISB        ;AND RETURN TO BASIC
2010 ;
2020 ;*************************************
2030 ;* ROUTINE TO CLEAR TBE BOTTOM LINE   *
2040 ;*************************************
2050 ;
2060 ERASE  LDA #<ERSMSG      ;ERASE SCREEN ACCEPT
2070        LDY #>ERSMSG      ;SET A,Y TO STRING
2080        JSR PRTSTR        ;AND PRINT IT
2090        RTS
2100 ;
2110 ;*************************************
2120 ;* MAIN LOGIC TO BANDLE ALL KEYBOARD  *
2130 ;* ENTRIES.                           *
2140 ;*************************************
2150 ;
2160 OK     JSR FDCRSR        ;OK - MOVE TO NEXT POSITION
2170        JSR HILGHT        ;HILIGHT NEW CRSR POSITION
2180 INPUT  JSR GET           ;GET A CHARACTER
2190        BEQ INPUT         ;NOTBING PRESSED - TRY AGAIN
2200        CMP #CD           ;CURSOR DOWN ?
2210        BEQ NXTFLD        ;YES - GO ONTO NEXT FIELD
2220        CMP #RTN          ;OR A RETURN ?
2230        BNE OK1           ;NO - GO ON
2240 NXTFLD JMP MAIN          ;YES - GO ON SCANNING
2250 OK1    CMP #HME          ;HOME ?
2260        BNE KYTEST        ;NO - GO ON
2270        JSR UNPLSH        ;UNBILIGHT PRESENT POSITION
2280        JMP BEGIN         ;YES - START AT PIRST PIELD
2290 KYTEST LDX #0            ;ZEROISE TABLE INDEX
2300 TEST   CMP NOGOOD,X      ;ANY GOOD ?
2310        BEQ INPUT         ;NO - GO BACK
2320        INX               ;BUMP TABLE INDEX
2330        LDY NOGOOD,X      ;END OF TABLE ?
2340        CPY #0
2350        BNE TEST          ;NO - CARRY ON
2360 OK2    CMP #SBPRTN       ;SHIPTED RETURN ?
2370        BNE OK3           ;NO - GO ON
2380        JMP ACCEPT        ;YES - GOTO SCREEN ACCEPT
2390 OK3    CMP #CU           ;CURSOR UP ?
2400        BNE OK4           ;NO - GO ON
2410        JSR SAVREG        ;YES - SAVE CURSOR POSITION
2420        JSR UNFLSB        ;UNBILIGHT CURSOR POSITION
2430        JSR SCANBD        ;SCAN BACK TO PREVIOUS FIELD
2440        BCS TOPSCR        ;ALREADY IN THE TOP PIELD!
2450        JMP OK            ;PINE - CARRY ON INPUTTING
2460 TOPSCR JSR RESREG        ;RESET CURSOR POSITION
2470        JSR HILGHT        ;RE-HILIGHT CURSOR POSITION
2480        JMP INPUT
2490 OK4    CMP #CLR          ;CLEAR SCREEN ?
2500        BNE OK5           ;NO - GO ON
2510        LDA #1            ;YES - SET ERRFLG
2520        STA ERRFLG        ;
2530        JMP FINISB        ;AND RETURN TO BASIC
2540 OK5    CMP #CL           ;CURSOR LEPT ?
2550        BEQ OK6           ;YES - GO ON
2560        CMP #DEL          ;DELETE ?
2570        BNE OK7           ;NO - GO ON
2580 OK6    PHA               ;SAVE CHARACTER
2590        JSR DECPTR        ;LOOK AT LEPT SIDE
2600        LDY #0            ;ZEROISE OPPSET
2610        LDA (POINTR),Y    ;GET CHARACTER
2620        CMP #OPENDL       ;IS IT AN OPEN DELIMITER ?
2630        BEQ RETURN        ;YES - GO ON
2640        JSR INCPTR        ;RESET POINTER
2650        PLA               ;NO - RETRIEVE CHARACTER
2660        CMP #DEL          ;WAS IT A DELETE ?
2670        BNE CRSRLP        ;NO - GO ON
2680        JMP DELETE        ;YES - GO TO IT
2690 CRSRLF JSR PRTCHR        ;NO - PRINT IT
2700        JMP INPUT         ;AND GO BACK
2710 RETURN PLA               ;FIX STACK
2720        JSR INCPTR        ;RESET POINTER
2730        JMP INPUT         ;YES - GO BACK
2740 OK7    CMP #INS          ;INSERT ?
2750        BNE OK8           ;NO - GO ON
2760        JMP INSERT        ;GO TO IT
2770 OK8    CMP #RUNSTP       ;RUN/STOP ?
2780        BNE OK9           ;NO - GO ON
2790        JSR UNFLSH        ;YES - UNHILIGHT CURSOR POSITION
2800        LDA #2            ;SET ERROR FLAG TO 2
2810        STA ERRFLG
2820        JMP FINISH        ;AND RETURN TO BASIC
2830 OK9    PHA               ;SAVE CHARACTER
2840 OK10   JSR SAVREG        ;SAVE CURSOR POSITION
2850        JSR INCPTR        ;MOVE TO RIGHT HAND SIDE
2860        LDY #0            ;ZEROISE OFFSET
2870        LDA (POINTR),Y    ;GET CHARACTER
2880        CMP #CLSEDL       ;IS IT A CLOSE DELIMITER ?
2890        BEQ NOPRT         ;YES - TAKE CARE OF IT
2900        JSR DECPTR        ;NO - RESET OLD CRSR POSITION
2910        PLA               ;RE-GET CHARCTER
2920        JSR PRTCHR        ;PRINT IT PROPERLY !
2930        JMP INPUT         ;AND CARRY ON
2940 ;
2950 NOPRT  JSR DECPTR        ;RESET POINTER
2960        JSR UNFLSH        ;UNHILIGHT CURSOR POSITION
2970        JSR SAVREG        ;SAVE REGISTERS
2980        PLA               ;CLEAN UP STACK
2990        JSR PRINT         ;PRINT IT
3000        JSR ESCAPE        ;DO AN ESCAPE
3010        JSR RESREG        ;RESET REGISTERS BACK ONTO
3020        JSR HILGHT        ;THE SAME SQUARE AND HILGHT
3030        JMP INPUT         ;IT AND CARRY ON
3040 ;
3050 ;*************************************
3060 ;* ROUTINE TO SET POINTER AND CURSOR *
3070 ;* TO THE HOME POSITION               *
3080 ;*************************************
3090 ;
3100 RESET  LDA #<SCREEN      ;GET LO BYTE OF SCREEN ADDRESS
3110        STA POINTR
3120        LDA #>SCREEN
3130        STA POINTR+1
3140        LDA #BME          ;LOAD A 'HOME'
3150        JSR PRINT         ;AND PRINT IT
3160        LDA #CD           ;LOAD A 'CURSOR DOWN'
3170        JSR PRINT         ;AND PRINT IT
3180        LDA #0            ;LOAD A ZERO
3190        STA ERRFLG        ;INITIALISE ERROR PLAG
3200        RTS               ;AND RETURN
3210 ;
3220 ;*************************************
3230 ;* ROUTINE TO SCAN PORWARD PROM THE   *
3240 ;* CURRENT CURSOR POSITION TO A FIELD.*
3250 ;* THE CARRY IS USED AS A FLAG STATING*
3260 ;* WHETBER OR NOT A FIELD WAS FOUND.  *
3270 ;*************************************
3280 ;
3290 SCANFD LDY #0            ;ZEROISE OFPSET
3300        LDA (POINTR),Y    ;GET A CHARACTER
3310        CMP #OPENDL       ;IS IT AN OPENING DELIMITER ?
3320        BEQ POUND         ;YES - RETURN
3330        JSR PDCRSR        ;NO - BUMP POINTER
3340        LDA #>ENDSCR      ;GET END OF SCREEN HI
3350        CMP POINTR+1      ;SAME AS POINTER HI ?
3360        BNE SCANFD        ;NO - KEEP ON SCANNING
3370        LDA #<ENDSCR      ;YES - GET END OP SCREEN LO
3380        CMP POINTR        ;SAME AS POINTER LO
3390        BNE SCANFD        ;NO - KEEP ON SCANNING
3400        SEC               ;YES - SET 'NOT FOUND' FLAG
3410        RTS               ;AND GO BACK
3420 POUND  CLC               ;SET FOUND PLAG
3430        RTS               ;AND GO BACK
3440 ;
3450 ;*************************************
3460 ;* ROUTINE TO SCAN BACKWARDS PROM THE *
3470 ;* CURRENT CURSOR POSITION TO A FIELD.*
3480 ;* THE CARRY IS USED AS A FLAG STATING*
3490 ;* WHETHER OR NOT A FIELD WAS FOUND.  *
3500 ;*************************************
3510 ;
3520 SCANBD LDA #0            ;GET A ZERO
3530        STA ERRFLG        ;ZEROISE FLAG
3540 BD1    LDY #0            ;ZEROISE OPFSET
3550        LDA (POINTR),Y    ;GET A CBARACTER
3560        CMP #OPENDL       ;IS IT AN OPENING DELIMITER ?
3570        BEQ FOUND1        ;YES - FOUND PIELD - GO ON
3580 BD2    JSR BKCRSR        ;NO - BUMP POINTER & CURSOR
3590        LDA #>SCREEN      ;GET START OP SCREEN HI
3600        CMP POINTR+1      ;SAME AS POINTER HI ?
3610        BNE BD1           ;NO - KEEP ON SCANNING
3620        LDA #<SCREEN      ;YES - GET START OF SCREEN LO
3630        CMP POINTR        ;SAME AS POINTER LO
3640        BNE BD1           ;NO KEEP ON SCANNING
3650        SEC               ;YES - SET 'NOT FOUND' PLAG
3660        RTS               ;AND GO BACK
3670 POUND1 DEC ERRFLG        ;DECREMENT FLAG
3680        BEQ POUND2        ;OK - WE'VE DONE IT
3690        INC ERRFLG        ;BUMP IT TO ZERO
3700        INC ERRFLG        ;THEN TO ONE
3710        JMP BD2           ;GO DO IT A SECOND TIME
3720 POUND2 CLC               ;SET 'FOUND FLAG'
3730        RTS               ;AND RETURN
3740 ;
3750 COUNT  .BYT 0            ;GENERAL COUNTER/STORE
3760 ;
3770 ;*************************************
3780 ;* ROUTINE TO HANDLE INSERTION KEY.   *
3790 ;* MOVES ALL CHARACTERS FROM CURSOR   *
3800 ;* POSITION TO THE END OF THE FIELD   *
3810 ;* UP ONE PLACE, THUS ERASING THE LAST*
3820 ;* CHARACTER IN THE FIELD. THEN A     *
3830 ;* SPACE IS PRINTED AT THE CURRENT    *
3840 ;* CURSOR POSITION.                   *
3850 ;*************************************
3860 ;
3870 INSERT LDY #0            ;ZEROISE OFPSET
3880        JSR INCPTR        ;MOVE ONTO NEXT SQUARE
3890 INS1   LDA (POINTR),Y    ;LOOK AT IT
```

```
3900        CMP #CLSEDL     ;END OF FIELD ?
3910        BEQ INS2        ;YES - GO ON
3920        INY             ;NO - BUMP INDEX
3930        BNE INS1        ;AND GO BACK
3940 INS2   STY COUNT       ;SAVE INDEX
3950        CPY #0          ;AT END OF FIELD ?
3960        BNE INS3        ;IP NOT AT END - GO ON
3970        JSR DECPTR      ;RESET POINTER
3980        JMP NOINST      ;AND GO ON
3990 INS3   JSR UNPLSH
4000        JSR DECPTR      ;RESET POINTER
4010        LDY COUNT       ;RESET INDEX
4020        INY
4030 INS4   DEY
4040        DEY
4050        LDA (POINTR),Y
4060        INY
4070        STA (POINTR),Y
4080        CPY #1          ;FINISHED ?
4090        BNE INS4        ;NO - GO BACK
4100        LDY #0          ;ZEROISE OFFSET
4110 NOINST LDA #160        ;LOAD A RVS SPACE
4120        STA (POINTR),Y  ;PUT IT BACK
4130        JMP INPUT       ;CARRY ON
4140 ;
4150 ;*************************************
4160 ;* ROUTINE TO HANDLE THE DELETE KEY.  *
4170 ;* MOVES ALL CHARACTERS PROM END OP   *
4180 ;* THE FIELD BACK TO THE CURRENT      *
4190 ;* CURSOR POSITION BACK ONE PLACE AND *
4200 ;* PRINTS A SPACE IN THE LAST POSITION*
4210 ;* OF THE FIELD. DOES NOT PUNCTION IN *
4220 ;* FIRST POSITION OF FIELD.           *
4230 ;*************************************
4240 ;
4250 DELETE JSR UNPLSH      ;UNHILIGHT CURSOR POSITION
4260        JSR INCPTR      ;MOVE ONTO NEXT SQUARE
4270        LDY #0          ;ZEROISE OPPSET
4280 DEL1   LDA (POINTR),Y  ;LOOK AT IT
4290        CMP #CLSEDL     ;END OF FIELD ?
4300        BEQ DEL2        ;YES - GO ON
4310        INY             ;NO - BUMP INDEX
4320        BNE DEL1        ;AND GO BACK
4330 DEL2   STY COUNT       ;SAVE INDEX
4340        JSR DECPTR      ;RESET POINTER
4350        JSR DECPTR      ;RESET POINTER
4360        LDY #1          ;SET OPPSET
4370 DEL3   LDA (POINTR),Y  ;GET A CHARACTER
4380        DEY
4390        STA (POINTR),Y  ;MOVE IT
4400        CPY COUNT       ;FINSHED ?
4410        BEQ DEL4        ;YES - GO ON
4420        INY             ;BUMP IT BACK
4430        INY             ;AND ONE MORE
4440        JMP DEL3        ;AND GO BACK
4450 DEL4   INC COUNT       ;FIX OFPSET
4460        LDY COUNT       ;GET IT
4470        LDA #32         ;LOAD A SPACE
4480        STA (POINTR),Y  ;PUT IT ON THE SCREEN
4490        LDA #CL         ;MOVE CURSOR, BACK ONE
4500        JSR PRTCHR
4510        JMP INPUT       ;AND GO BACK
4520 ;
4530 ;*************************************
4540 ;* ROUTINE TO PRINT A STRING WHICH IS *
4550 ;* STORED AT THE ADDRESS IN THE       *
4560 ;* ACCUMULATOR AND THE Y REGISTER.    *
4570 ;* PRINTS MESSAGE ON BOTTOM LINE AT   *
4580 ;* 'MSGPN' WHICH IS DEPENDENT ON THE  *
4590 ;* SCREEN WIDTH, IN ORDER TO CENTER   *
4600 ;* THE MESSAGE.                       *
4610 ;*************************************
4620 ;
4630 PRTSTR STA POINTR      ;PUT A,Y IN POINTER
4640        STY POINTR+1
4650        LDY #0          ;ZEROISE OPPSET
4660        LDA #23         ;SET PRINT AT BOTTOM LINE
4670        STA CRSRLN
4680        LDA #RTN        ;PRINT A RETURN
4690        JSR PRINT
4700        LDA #MSGPN      ;SET COLUMN
4710        STA CRSRPN
4720 PRLOOP LDA (POINTR),Y  ;GET A CHARACTER
4730        BEQ GOBACK      ;IP NULL - END OP STRING
4740        JSR PRINT       ;PRINT THE CHARACTER
4750        JSR INCPTR      ;BUMP THE POINTER
4760        BNE PRLOOP      ;AND GO ONTO THE NEXT
4770 GOBACK JMP UPDATE      ;PIX POINTER & RTS
4780 ;
4790 ;*************************************
4800 ;* ROUTINE TO INCREMENT THE 16 BIT    *
4810 ;* POINTER 'POINTR' BY ONE.           *
4820 ;*************************************
4830 ;
4840 INCPTR INC POINTR      ;INCREMENT LO BYTE
4850        BNE NOINC       ;NOT ZERO - DON'T BUMP HI BYTE
4860        INC POINTR+1    ;BUMP POINTER HI
4870 NOINC  RTS
4880 ;
4890 ;*************************************
4900 ;* ROUTINE TO DECREMENT THE 16 BIT    *
4910 ;* POINTER 'POINTR' BY ONE.           *
4920 ;*************************************
4930 ;
4940 DECPTR LDA POINTR      ;GET POINTER LO
4950        BNE NODEC       ;NOT ZERO - DON'T BUMP HI BYTE
4960        DEC POINTR+1    ;DECREMENT THE HI BYTE
4970 NODEC  DEC POINTR      ;DECREMENT THE LO BYTE
4980        RTS
4990 ;
5000 ;*************************************
5010 ;* ROUTINE TO INCREMENT THE 16 BIT    *
5020 ;* POINTER 'ARRPTR' BY ONE. THIS      *
5030 ;* POINTER IS USED TO FIND THE SC$    *
5040 ;* ARRAY.                             *
5050 ;*************************************
5060 ;
5070 ARRINC INC ARRPTR      ;INCREMENT LO BYTE
5080        BNE NOINC       ;NOT ZERO - DON'T BUMP HI BYTE
5090        INC ARRPTR+1    ;BUMP POINTER HI
5100        RTS
5110 ;
5120 ;*************************************
5130 ;* ROUTINE TO SIMULATE PRINTING A     *
5140 ;* CURSOR RIGHT CHARACTER. THIS IS    *
5150 ;* USED INSTEAD OF ACTUALLY PRINTING  *
5160 ;* A CURSOR RIGHT, BECAUSE IT IS A    *
5170 ;* GREAT DEAL PASTER!                 *
5180 ;*************************************
5190 ;
5200 PDCRSR JSR INCPTR      ;BUMP POINTER ON ONE
5210        INC CRSRPN      ;INCREMENT CURSOR POSITION
5220        LDA #LINLEN     ;LOAD ACCUMULATOR WITH LINELENGTH
5230        CMP CRSRPN      ;SAME AS CURSOR POSITION ?
5240        BNE CRSROK      ;NO - GO ON
5250        CLC             ;YES - ADD LINELENGTH TO CRSR LINEPTR
5260        ADC CRSRLO      ;ADD LO BYTE
5270        STA CRSRLO      ;STORE IT
5280        LDA #0          ;GET 0
5290        STA CRSRPN      ;RESET CURSOR POSITION
5300        ADC CRSRHI      ;ADD HI BYTE + CARRY
5310        STA CRSRHI      ;STORE IT
5320        INC CRSRLN      ;BUMP CURRENT LINE
5330 CRSROK RTS             ;AND GO BACK
5340 ;
5350 ;*************************************
5360 ;* ROUTINE TO SIMULATE PRINTING A     *
5370 ;* CURSOR LEFT CHARACTER. THIS IS     *
5380 ;* USED INSTEAD OF ACTUALLY PRINTING  *
5390 ;* A CURSOR LEPT AGAIN BECAUSE IT IS A*
5400 ;* GREAT DEAL PASTER!                 *
5410 ;*************************************
5420 ;
5430 BKCRSR JSR DECPTR      ;BUMP POINTER DOWN ONE
5440        DEC CRSRPN      ;DECREMENT CURSOR POSITION
5450        BPL CRSROK      ;IF STILL POSITIVE THEN OK!
5460        LDA #LINLEN-1   ;IT'S BELOW ZERO - SET IT TO LINELENGTH-1
5470        STA CRSRPN      ;
5480        SEC             ;SUBTRACT LINELENGTH PROM CRSR LINEPTR
5490        LDA CRSRLO      ;GET LO BYTE
5500        SBC #LINLEN     ;SUBTRACT LENGTH OF LINE
5510        STA CRSRLO      ;STORE RESULT
5520        LDA CRSRHI      ;GET HI BYTE
5530        SBC #0          ;SUBTRACT 'BORROW' (IF ANY)
5540        STA CRSRHI
5550        DEC CRSRLN
5560        RTS             ;AND GO BACK
5570 ;
5580 ;*************************************
5590 ;* ROUTINE TO HILIGBT THE CURRENT     *
5600 ;* CURSOR POSITION.                   *
5610 ;*************************************
5620 ;
5630 HILGHT LDY CRSRPN      ;LOAD OPPSET
5640        LDA (CRSRLO),Y  ;GET CHR AT CURSOR POSITION
5650        ORA #$80        ;BILGHT IT
5660        STA (CRSRLO),Y  ;PUT CHR BACK AT POSITION
5670        RTS
5680 ;
```

```
5690 ;**************************************
5700 ;* ROUTINE TO UN-HILIGHT THE CURRENT  *
5710 ;* CURSOR POSITION.                   *
5720 ;**************************************
5730 ;
5740 UNFLSH LDY CRSRPN      ;LOAD OFFSET
5750        LDA (CRSRLO),Y  ;GET CHR AT CURSOR POSITION
5760        AND #$7F        ;PUT IT OUT OP REVERSE
5770        STA (CRSRLO),Y  ;PUT CHR BACK AT POSITION
5780        RTS
5790 ;
5800 ;**************************************
5810 ;* ROUTINE TO OUTPUT CHARACTER TO     *
5820 ;* SCREEN. THE JSR ESCAPE IS USED     *
5830 ;* RATHER THAN PRINTING AN ESCAPE     *
5840 ;* CHARACTER, BECAUSE THIS CHARACTER  *
5850 ;* IS NOT AVAILABLE ON THE 40 COLUMN  *
5860 ;* PET, AND SO THIS ROUTINE SIMULATES *
5870 ;* IT.                               *
5880 ;**************************************
5890 ;
5900 PRTCHR PHA             ;SAVE CHARACTER TO BE PRINTED
5910        JSR UNFLSH      ;UN-HIGHLIGHT CURSOR POSITION
5920        PLA             ;GET CHARACTER TO PRINT
5930        JSR PRINT       ;PRINT IT
5940        JSR ESCAPE      ;DO AN ESCAPE
5950        JSR HILGHT      ;HIGHLIGHT NEW POSITION
5960 UPDATE CLC             ;UPDATE POINTERS
5970        LDA CRSRPN      ;GET CURSOR POSITION
5980        ADC CRSRLO      ;ADD LO BYTE OF START OF LINE
5990        STA POINTR      ;STORE IT
6000        LDA #0          ;ADD CARRY TO HIGH BYTE
6010        ADC CRSRHI      ;
6020        STA POINTR+1    ;STORE IT
6030        RTS             ;AND RETURN
6040 ;
6050 ;**************************************
6060 ;* ROUTINE TO SIMILATE PRINTING AN    *
6070 ;* ESCAPE CHARACTER, AVAILABLE ON THE *
6080 ;* 80 COLUMN PET. THIS CHARACTER      *
6090 ;* NULLIFIES THE AFFECT OF EITHER     *
6100 ;* PRESSING THE INSERTION KEY, OR     *
6110 ;* WHEN INSIDE QUOTES, I.E TO STOP    *
6120 ;* CONTROL CHARACTERS BEING DISPLAYED *
6130 ;* LITERALLY.                        *
6140 ;**************************************
6150 ;
6160 ESCAPE LDA #0          ;NULLIFY QUOTE MODE
6170        STA QTEFLG
6180        STA NUMINS      ;AND SAY - NO INSERTS
6190        LDA #OFFRVS     ;LOAD AN OFF REVERSE
6200        JMP PRINT       ;PRINT IT AND RETURN
6210 ;
6220 REGSAV .BYT 0,0,0,0,0,0 ;STORAGE AREA
6230 ;
6240 ;**************************************
6250 ;* ROUTINE TO SAVE CURRENT CURSOR     *
6260 ;* POSITION AND THE CONTENTS OF THE   *
6270 ;* POINTER 'POINTR'.                  *
6280 ;**************************************
6290 ;
6300 SAVREG LDA CRSRLO      ;GET CURSOR LINE LO
6310        STA REGSAV      ;SAVE IT
6320        LDA CRSRHI      ;GET CURSOR LINE HI
6330        STA REGSAV+1    ;SAVE IT
6340        LDA CRSRPN      ;GET CURSOR POSITION
6350        STA REGSAV+2    ;SAVE IT
6360        LDA CRSRLN      ;GET CURRENT LINE
6370        STA REGSAV+3    ;SAVE IT
6380        LDA POINTR      ;GET POINTER LO
6390        STA REGSAV+4    ;SAVE IT
6400        LDA POINTR+1    ;GET POINTER HI
6410        STA REGSAV+5    ;SAVE IT
6420        RTS             ;AND GO BACK
6430 ;
6440 ;**************************************
6450 ;* ROUTINE TO RESET CURRENT CURSOR    *
6460 ;* POSITION AND THE POINTER 'POINTR'  *
6470 ;* TO THE SAVED VALUES.               *
6480 ;**************************************
6490 ;
6500 RESREG LDA REGSAV      ;GET CURSOR LINE LO
6510        STA CRSRLO      ;RESET IT
6520        LDA REGSAV+1    ;GET CURSOR LINE HI
6530        STA CRSRHI      ;RESET IT
6540        LDA REGSAV+2    ;GET CURSOR POSITION
6550        STA CRSRPN      ;RESET IT
6560        LDA REGSAV+3    ;GET CURRENT LINE
6570        STA CRSRLN      ;RESET IT
6580        LDA REGSAV+4    ;GET POINTER LO
6590        STA POINTR      ;RESET IT
6600        LDA REGSAV+5    ;GET POINTER HI
6610        STA POINTR+1    ;RESET IT
6620        RTS             ;AND GO BACK
6630 ;
6640 ;**************************************
6650 ;* CODE TO FIND SC$ ARRAY AND READ    *
6660 ;* THE CONTENTS OF THE FIELDS INTO    *
6670 ;* THE ARRAY, USING THE LENGTHS OF    *
6680 ;* ARRAY ELEMENTS AS THE NUMBER OF    *
6690 ;* CHARACTERS TO BE READ FROM THE     *
6700 ;* START OP THE FIELD. IT WILL ABORT  *
6710 ;* IF THE ARRAY DOES NOT EXIST, AN    *
6720 ;* ELEMENT OF THE ARRAY IS NULL, THE  *
6730 ;* END OF THE SCREEN IS REACHED, OR   *
6740 ;* THE LAST ELEMENT OF THE ARRAY IS   *
6750 ;* DONE.                              *
6760 ;**************************************
6770 ;
6780 GETSTR LDA ARRTAB      ;SET ARRPTR
6790        STA ARRPTR      ;TO FIND SC$ ARRAY
6800        LDA ARRTAB+1
6810        STA ARRPTR+1
6820        JSR RESET
6830        LDY #0          ;ZEROISE OFFSET
6840 LOOP   LDA (ARRPTR),Y  ;LOAD VAR1
6850        CMP #'S         ;IS IT AN 'S' ?
6860        BNE INC7        ;NO - GO ON
6870        JSR ARRINC      ;INCREMENT ARRAY PTR
6880        LDA (ARRPTR),Y  ;LOAD VAR2
6890        CMP #195        ;SHIFTED 'C' ?
6900        BNE INC6        ;NO - GO ON
6910        JMP FOUND4      ;FOUND IT - GO ON
6920 INC7   JSR ARRINC      ;MOVE ON A BYTE
6930 INC6   JSR ARRINC      ;AND ANOTHER
6940        LDA (ARRPTR),Y  ;GET PTRLO TO NEXT ARRAY
6950        STA REGSAV      ;SAVE IT
6960        JSR ARRINC      ;MOVE ONTO NEXT BYTE
6970        LDA (ARRPTR),Y  ;GET PTRHI TO NEXT ARRAY
6980        STA REGSAV+1    ;SAVE IT
6990        LDA ARRPTR      ;GET LO BYTE
7000        CLC
7010        ADC REGSAV      ;ADD OPFSET LO
7020        STA ARRPTR      ;SET ARRAY PTR LO
7030        LDA ARRPTR+1    ;GET HI BYTE
7040        ADC REGSAV+1    ;ADD OPPSET HI
7050        STA ARRPTR+1    ;SET ARRAY PTR HI
7060        SEC
7070        LDA ARRPTR      ;SUBTRACT 3
7080        SBC #3
7090        STA ARRPTR
7100        LDA ARRPTR+1
7110        SBC #0
7120        STA ARRPTR+1
7130 ;
7140        LDA ARRPTR+1    ;END OP ARRAYS ?
7150        CMP ARREND+1
7160        BCC LOOP        ;NO - KEEP LOOKING
7170        BNE EXIT        ;YES - EXIT
7180        LDA ARRPTR      ;LO BYTE ?
7190        CMP ARREND
7200        BCC LOOP        ;NO - KEEP LOOKING
7210 EXIT   JMP FINISH      ;AND RETURN
7220 ;
7230 FOUND4 JSR ARRINC      ;MOVE ONTO LO
7240        JSR ARRINC      ;& HI OF NEXT ARRAY
7250        JSR ARRINC      ;NUM DIMENSIONS
7260        LDA (ARRPTR),Y  ;GET IT
7270        CMP #1          ;SINGLE DIMENSION ?
7280        BEQ FOUND5      ;YES - GO ON
7290        JMP FINISH      ;NO - RETURN
7300 FOUND5 JSR ARRINC      ;HI OF ARRAY SIZE
7310        JSR ARRINC      ;LO OF ARRAY SIZE
7320        LDA (ARRPTR),Y  ;GET IT
7330        STA COUNT       ;SAVE IT
7340 GT3    JSR ARRINC      ;ONTO ARRAY ELEMENT
7350        LDA (ARRPTR),Y  ;GET LENGTH OF IT
7360        BNE NOTNUL      ;NOT NULL STRING - GO ON
7370        JMP FINISH      ;NUL STRING - STOP
7380 NOTNUL TAX             ;SAVE IT
7390        JSR ARRINC      ;MOVE ON ONE
7400        LDA (ARRPTR),Y  ;GET LO
7410        STA STRPTR      ;SET STRPTR LO
7420        JSR ARRINC      ;MOVE ON ONE
7430        LDA (ARRPTR),Y  ;GET HI
7440        STA STRPTR+1    ;SET STRPTR HI
7450        LDA STRPTR
7460        SEC
7470        SBC #1
7480        STA STRPTR
7490        LDA STRPTR+1
7500        SBC #0
7510        STA STRPTR+1
7520        JSR SCANFD      ;FIND PIELD ON SCREEN
7530        BCC PILL        ;FOUND ONE - GO ON
7540        JMP PINISH      ;NOT POUND - RETURN
7550 ;
7560 PILL   TXA             ;GET LENGTH
7570        TAY             ;SET OFPSET
7580 FILL1  LDA (POINTR),Y
7590 CONV1  PHA             ;SAVE BIT 6
7600        ASL A
7610        ASL A
7620        PLA
7630        PHP
7640        AND #$3F
7650        CMP #$20
7660        BCS CONV2
7670        ORA #$40
7680 CONV2  PLP
7690        BCC DONE
7700        ORA #$80
7710 DONE   STA (STRPTR),Y
7720        DEY
7730        BNE PILL1
7740        DEC COUNT       ;MORE STRINGS ?
7750        BNE MOREFD      ;YES - GO ON
7760        JMP PINISH      ;NO - RETURN
7770 MOREFD JSR FDCRSR      ;BUMP CRSR & POINTER
7780        JMP GT3
7790 ;
7800 ACCMSG .BYT RVS,'PRESS SHIFT RETURN TO'
7810        .BYT ' ACCEPT SCREEN.',OFFRVS,0
7820 ;
7830 ERSMSG .BYT RVS,'                    '
7840        .BYT '                ',OFFRVS,0
7850 ;
7860 NOGOOD .BYT RVS,OFFRVS,ESC
7870 ; -------------  PROTECT FROM NASTIES ! --------------
7880        .BYT 25,153,14,142,131,7,21,149,22,158,15,143
7890 ; ----------------------------------------------------
7900        .BYT SHFESC,TAB,SHPTAB
7910        .BYT OPENDL,CLSEDL,0
7920 ;
7938 .END

READY.
```

EXAMPLE PROGRAM

```
1000 poke53,112:clr:rem lower top of memo
     ry and clr to fix pointers
1010 dim ln(8),sc$(8):rem dimension array
     s (length, field contents)
1015 :
1020 rem data <field prompt>, <field leng
     th>
```

```
1030 data "Company",20,"Address1",20,"Add
     ress2",15,"Address3",12
1040 data "Address4",10,"Contact",20,"Tel
     .",15,"Comments",28
1045 :
1050 fori=0to7:rem loop to read data stat
     ements into arrays
1060 readpr$(i),ln(i):rem read one prompt
      and one length into array
1070 nexti:rem and continue with loop
1080 :
1090 print"":poke59468,14:rem put into
     lower case on any pet
1100 :
1110 rem create string of spaces to fill
     a null record
1120 sp$="
                                        "
1130 :
1140 fori=0to7:rem loop to fill each elem
     ent with spaces
1150 sc$(i)=left$(sp$,ln(i)):rem sc$(i)=
     ln(i) spaces
1160 nexti:rem and continue with loop
1170 :
1180 pokel44,peek(144)+3
1200 gosub2000:rem display record
1210 :
1220 sys31488:rem enter editor
1230 :
1240 on peek(0)+1 gotol300,1200,3000
1250 rem peek(0)=0 'ok'
1260 rem peek(0)=1 came back to basic via
      clear screen
1270 rem peek(0)=2 came back to basic via
      run/stop (i.e help/abort) function
1280 :
1300 print"Line. You entered :-"
1310 fori=0to7:rem loop to display conten
     ts of array
1320 print"sc$("mid$(str$(i),2)")="chr$(3
     4)sc$(i)chr$(34):rem display element
1330 nexti:rem contimnue with loop
1335 pokel44,peek(144)-3
1340 end:rem end of program
1350 :
2000 a$="fixed screen environment":print"
     ";:a=int((78-len(a$))/2)
2001 printleft$(sp$,a)a$left$(sp$,a)
2002 a$="Press stop key for help":print"
     ";
2003 a=int((78-len(a$))/2)
2004 printleft$(sp$,a)a$left$(sp$,a)"";
2005 print"";:rem routine to displ
     ay record on the screen
2010 fori=0to7
2020 ifi=lori=5thenprint"";:rem space ou
     t fields
2030 printpr$(i)tab(9)"<"sc$(i)">"
2040 next
2050 return
2060 :
3000 print"Help Function would operate h
     ere.":rem help facility
3005 gosub3500
3010 geta$:ifa$<>""then3010
3020 print" ress a key
3030 geta$:ifa$=""then3030
3040 print"":gotol200
3500 p=1025
3505 lk=peek(p)+peek(p+1)*256:p=p+2:iflk
     =0thenreturn
3510 ln=peek(p)+peek(p+1)*256:ifln=4000
     then3600
3520 p=lk:goto3505
3600 p=p+4
3610 ifpeek(p)<>0thenprintchr$(peek(p));:
     p=p+1:goto3610
3620 print:p=lk:lk=peek(p)+peek(p+1)*256:
     p=p+2:iflk<>0then3600
3630 return
3990 rem
4000 rem"******  Screen Editor Rules  ***
     ***
4010 rem"-------------------------------
     ---
4020 rem"Within each field, insert, delet
     e, and ascii keys, work in the
4030 rem"usual way.
4040 rem"Cursor down (return works in the
      same way) moves onto the next field
4050 rem"if there is one, or moves to the
      "screen accept position". This
4060 rem"means that the operator can eith
     er accept the entry, by entering
4065 rem"a shift/return as shown on the b
     ottom line of the screen, or continu
     e
4070 rem"editing the screen by typing a c
     ursor up, to move up into the last
4080 rem"field, or home to move to the fi
     rst field, or run/stop to jump to
4090 rem"the help/abort facility, or clr/
     home to reset all of the fields to
4100 rem"their default values.
4110 rem"A shifted return can be pressed
     at any time when in the environment
4120 rem"to move directly to the screen a
     ccept position.
4130 rem"The stop key provides a help/abo
     rt activity.
```

Communications is the Name of the Game

Mike Shingler
Kingston Computers

For the micro-computer business, 1981 must be 'The year of communications'. Without doubt the ability to link systems together is of significant interest to a great many users.

Until recently communications methods have been primarily concerned with Mainframe and Mini markets, which have inevitably resulted in fairly complex, expensive hardware and software techniques. Although the same principles apply one can now achieve effective communications between Micro's and many other devices such as Mainframes, Mini and specialist measuring/monitoring devices.

Possibly one of the most versatile of these communication devices is the Kingston NETKIT, a hardware/firmware device designed and developed by Kingston for the Commodore PET; transforming it into a more powerful and easily controllable beast at very low cost.

Industry, commerce and education are now discovering numerous applications with NETKIT which have been previously impossible to achieve, included among these are:-

1. Production Control: By linking a PET to a paper tape punch/reader NC machine tapes are able to be quickly analysed to determine machine cycle time and to modify tapes if required.

2. Education: By linking PET's in schools to a central Mini or Mainframe file and program sharing for educational purposes are now possible.

3. Instrumentation Control: Allowing the PET fitted with a NETKIT to analyse various forms of data from differing types of monitoring and sampling devices both quickly and effortlessly.

4. Mainframe and Mini: With NETKIT the PET can act as a terminal to a Mainframe or Mini for normal terminal type requirements.

5. Test, Maintenance: The PET upgraded with a NETKIT can be used as a testing device for other devices such as printers.

For these and many other applications, some of which are just around the corner; for example Electronic Mail, which will save organisations both time and money against current manual methods, 1981 is going to be 'The year of Communications'.

Editors note:
I asked Mike to write a short article about communications on the PET. I get the feeling however that he has biased his text towards the NETKIT! The NETKIT is a Commodore Approved Product and costs £135.00+VAT from your local dealer.